University of Nottingham
UK | CHINA | MALAYSIA

EVENTMAP

12-15 Dec 2019

MISTA 2019 Proceedings

12–15 Dec 2019 | Ningbo, China

ISSN: 2305-249X

MISTA 2019

Proceedings of the

9th Multidisciplinary International Conference on Scheduling: Theory and Applications

12 – 25 December 2019
Ningbo, China

Edited by
Ruibin Bai, University of Nottingham Ningbo China
Zhi-Long Chen, University of Maryland
Li Jiawei, University of Nottingham Ningbo China
Graham Kendall, University of Nottingham, Malaysia and UK
Barry McCollum, Queens University Belfast, UK

MISTA 2019 Conference Program Committee

Abdullah, Salwani (Universiti Kebangsaan Malaysia)
Ayob, Masri (Universiti Kebangsaan Malaysia)
Bai, Ruibin (University of Nottingham Ningbo Campus)
Barták, Roman (Charles University)
Berghman, Lotte (Toulouse Business School)
Briand, Cyril (LAAS-CNRS)
Bulbul, Kerem (Sabanci University)
Ceschia, Sara (DIEGM, University of Udine)
Chan, Hing Kai (University of Nottingham)
Che, Ada (Northwestern Polytechnical University)
Chen, Zhi-Long (University of Maryland)
Chen, Xin (Liaoning University of Technology)
Chong, Siang Yew (School of Computer Science, University of Nottingham, Malaysia Campus)
Drake, John H. (Queen Mary University of London)
Drozdowski, Maciej (Poznan University of Technology)
Feng, Li (Liaoning Key Laboratory of Manufacturing System and Logistics, Institute of Industrial Engineering & Logistics Optimization)
Gao, Liang (Huazhong Univ. of Sci. & Tech.)
Geismar, Neil (Texas A&M University)
Goossens, Dries (Ghent University)
Gunawan, Aldy (Singapore Management University)
Hao, Jin-Kao (University of Angers)
Herrmann, Jeffrey W. (University of Maryland)
Huo, Yumei (City University of New York)
Kaminsky, Phil (University of California, Berkeley)
Kendall, Graham (University of Nottingham, Malaysia and UK)
Kingston, Dr. Jeffrey H. (The University of Sydney)
Knust, Sigrid (University of Osnabrueck)
Kubiak, Wieslaw (MUN)
Lee, Kangbok (POSTECH)
Lee, Lai Soon (Universiti Putra Malaysia)
Li, Jiawei (University of Nottingham Ningbo Campus)
Li, Chung-Lun (The Hong Kong Polytechnic University)
Liu, Zhixin (University of Michigan)
Luo, Wenchang (School of Mathematics and Statistics, Ningbo University)
McCollum, Barry (Queen's University Belfast)
Moench, Lars (University of Hagen)
Özcan, Ender (University of Nottingham)
Posner, Marc (The Ohio State University)
Qi, Xiangtong (The Hong Kong University of Science and Technology)
Qu, Rong (University of Nottingham)
Ribeiro, Celso (Universidade Federal Fluminense)
Rossi, Andre (Universite Paris-Dauphine)
Rudová, Hana (Masaryk University)
Sagir, Mujgan (Eskisehir Osmangazi University)
Schaerf, Andrea (University of Udine)
Seow, Hsin-Vonn (The University of Nottingham Malaysia Campus)
Shi, Xuhua (Ningbo University)
Sriskandarajah, Chelliah (Texas A&M University)

Subramanian, Nachiappan (University of Sussex)
Sucha, Premysl (Department of Control Engineering, Czech Technical University)
Tan, Zhen (University of Nottingham)
Thompson, Jonathan (Cardiff University)
Trick, Michael (Carnegie Mellon University)
Urrutia, Sebastián (Universidade Federal de Minas Gerais)
Vansteenwegen, Pieter (KU Leuven Mobility Research Centre)
Vincent, T'Kindt (Laboratoire d'Informatique)
Wang, Junqiang (Northwestern Polytechnical University)
Wang, Ling (Tsinghua University)
Wauters, Tony (KU Leuven, CODeS)
Xu, Zhou (The Hong Kong Polytechnic University)
Yu, Vincent F. (National Taiwan University of Science and Technology)
Yugma, Claude (Ecole des Mines de Saint-Etienne)
Zhang, Guochuan (Zhejiang University)
Zhang, Xiandong (Fudan University)
Zhao, Hairong (Purdue University Northwest)
Zinder, Yakov (University of Technology, Sydney)

Acknowledgements

This conference would not have been possible without the assistance of a great many people.

Any scientific conference is underpinned by the quality of the papers that it publishes. This is largely the responsibility of the Program Committee, who give up their time and expertise to assist us. MISTA is fortunate enough to have many of the world's scheduling experts who help us by serving on the Program Committee. We are extremely grateful for the time and effort that they devote to making MISTA the success it is.

MISTA 2019 had five plenary speakers (**Tan Kay Chen**, **Nicholas G. Hall**, **José F. Oliveira**, **Lixin Tang** and **Ling Wang**). We are extremely grateful to these five world leading researchers, who gave their time to present to the MISTA delegates. We know that everybody would have got a lot from their expertise and insights.

This year we were delighted to have three tutorials (*Appointment Scheduling* (**Guahua Wan**), *Operations Scheduling with Multitasking* (**Chung-Lun Li**), *Stabilizing the grand coalition for cooperative games: A unified framework demonstrated by scheduling games* (**Xiangtong Qi**)). We express our thanks to these three tutorial speakers for making their time and expertise available to the conference.

We greatly appreciate the support that we have received from EventMap Ltd., which have supported the conference once again. We are also grateful to the University of Nottingham (in China, Malaysia and the UK) for their continued support of this conference series.

We would also like to thank the International Journal of Production Research by allowing us to guest edit a special issue of the journal which is associated with the conference. This certainly adds to the conference and the post-conference opportunities.

Our sincere thanks must also go to a dedicated and hardworking local team (**Jianfeng Ren**, **Chris Roadknight**, **Zheng Lu**, **Sheung Hung Poon**, **Joyce Jin**, **Rong Qu**, **Junqiang Wang**, **Lining Xing**, **Qingguo Bai**, **Danyu Bai**, **Jane Wang** and **Kelly Fu**). Without their support, MISTA would not happen, and certainly not happen as smoothly as it would without their help. We can only imagine how much work went into planning and preparing for this conference.

As in previous years, the conference owes a great deal to **Debbie Pitchfork**. She has worked tirelessly since performing similar tasks for MISTA 2009, 2011, 2013, 2017 and 2017. If it were not for Debbie this conference would not have taken place. Thank you, from all who are involved in MISTA 2019, whether as part of the organisational team, or the delegates.

Table of Contents

Plenary Presentations

Chen T.K. *Advances in Evolutionary Multitasking for Vehicle Routing* 10

Hall N. G. *A New Design for Tournaments* 11

Oliveira J. F. *Tackling uncertainty in combinatorial optimization problems: using metaheuristics to efficiently co-generate scenarios and solutions* 12

Tang L. *Data Analytics and Optimization for Steel Industry* 13

Wang L. *Data-Driven Intelligent Optimization Scheduling* 14

Papers

Tu C., Bai R., Lu Z., Aickelin U., Ge P. and Zhao J. *Learning Regular Expressions for Interpretable Medical Text Classification Using a Pool-based Simulated Annealing and Word-vector Models* 16

Aronshtam L., Ilani H. and Shufan E. *Satisfying Allocation of Plots to Tenants* 30

Vacher B., Jouglet A., Nace D., Pietrowicz S. and Bouznif M. *Aggregating partial orders - a case study in the internal logistics sector* 42

Pham T-S., Rousseau L-M. and De Causmaecker P. *Radiotherapy scheduling problem* 53

Nurit B., Baruch M. and Dana S. *Lot Scheduling Problems on Identical Parallel Machines* 65

Lin Y., Wu W., Hu Y. and Yagiura M. *Heuristics for the Min-Max Regret Shortest Path Problem under Interval Costs* 76

Luo W., Fang K. and Jin M. *Rescheduling for new orders on a single machine with rejection* 100

Zhang X., Bai D., Lin W-C. and Wu C-C. *Two rescheduling problems with allowing for the unexpected arrival of new jobs* 109

Ikar E., Grinshpoun T., Ilani H. and Shufan E. *Reentrant Flow Shop with Identical Jobs and Unit Processing Times* 125

Böðvarsdóttir E. B., Smet P., Vanden Berghe G. and Stidsen T. *A modeling methodology to support nurse rostering practitioners* 141

Ding S., Cui T. and Bai R. *Production Planning based on Supply Chain Volatility Clustering: A Machine Learning Approach* 156

Chen P., Xu M., Tan J. and Zhang S. *The Electric Vehicle Routing Problem with Time Windows and Mobile Recharging* 169

Çalık H., Hallaert A., Karami F. and Vanden Berghe G. *The video sequencing problem* 186

Karami F., Vancroonenburg W. and Vanden Berghe G. *Facilitating same-day delivery through conveyor operations modeling and optimization in distribution centers* 197

Schulte J. and Nissen V. *Holistic Workforce Planning: Integrating Staffing, Shift Design and Scheduling using Evolutionary Bilevel Optimization* 209

Li X. and Ada C. *Branch-and-price-and-cut Algorithm of Selective Pickup and Delivery Problem with Loading Cost and Time Window* 224

Feng S., Xiong Q. and Zhao X. *An Improved Opposition-based Differential Evolution with Gaussian Perturbation* 249

Wang Y. and Deng Q. *A Heuristic Algorithm To Compute A Subgraph For TSP Based On Frequency Quadrilaterals* 257

Tian Y., Xiong T., Liu Z., Deng P. and Wan L. *Applying resource-leveling strategy for solving Multi-Skill Resource-Constrained Project Scheduling Problem* 271

Ji M., Zhang X., Peng J., Huang Y. and Xiang W. *Flexible Outpatient Appointment Decision Considering Walk-in Patient with Dense Arrival* 285

Baruch M. and Dana S. *Single Machine Scheduling with Non-availability Interval and Optional Job Rejection* 293

Kyngäs N., Nurmi K. and Goossens D. *The General Task-based Shift Generation Problem: Formulation and Benchmarks* 301

Peng C., Li Z., Chen Q. and Liu J. *Two algorithms for the Flexible Job Shop Scheduling Problem with limited transportation capacity* 320

Zhong W., Ding H. and Yu Y. *Scheduling problems with assembly and delivery operations in Alibaba's Hema Store* 328

Nugroho H., Kusuma D. W. and Biyanto T. R. *Optimization of Heat Exchanger Cleaning Schedule for Natural Gas Liquefaction Process* 340

Zhong H., Liu J., Chen Q. and Li Z. *Integrating load based controlled release with priority dispatching for dynamic flexible assembly job shop with alternative process plans* 350

Xu D., Li K., Bai Q. and Cui L. *A Multi-Commodity Unpaired Pickup and Delivery Vehicle Routing Problem with Split Loads and Unloads* 360

Bassimir B. and Wanka R. *Robustness Approaches for the Examination Timetabling Problem under Data Uncertainty* 381

Fuchigami H. Y. *A computational experimentation for the flow shop scheduling problem with due date assignment* 396

Thanos E., Van den Bossche T., Santos H. G., Vancroonenburg W. and Vanden Berghe G. *Feature-based tuning of a Tactical Berth Allocation heuristic* 410

Lu X-Q., Wu C-X.., Zhang M-X. and Zheng Y-J. *Water Wave Optimization for Integrated Bank Customer Service Scheduling* 429

Huang L., Ye C., Chen K., Zhao T. and Qu C. *Research on Dynamic Balance of Mixed-model Assembly Line Based on Hybrid Genetic-Particle Swarm Optimization* 445

Xu S. and Chan H. K. *Forecasting port container throughput by "Ridge Regression - Seasonal Autoregression Integrated Moving Average - Support Vector Regression" approach* 463

Yang Q., Tang S. and Liu Z. *Integrated order batching and vehicle routing in cigarette distribution* 470

Abstracts

Hermans B., Matuschke J. and Leus R. *Exact and approximation algorithms for the expanding search problem* 481

Itoi M., Wu W. and Ikegami A. *Resource Balancing for Scheduling Problems* 484

Grigoreva N. *Scheduling Tasks with Release and Delivery Times on Parallel Processors* ... 487

Zhou S. and Yue Q. *Sequencing and Scheduling Appointments for Multi-stage Service Systems with Stochastic Service Durations and No-shows* 491

Yi X. and Goossens D. *Dealing with uncertainty in time-relaxed round robin sports timetabling* 495

Zhang Z., Pi B., Song Y., Xing L. and Chen Y. *A Novel Artificial and Heuristic Hybrid Algorithm Method for Satellite Data Downlink Scheduling Problem with Malfunctions* .499

Stidsen T. *EURO-2019 conference scheduling - Use of session patterns* 500

Yang F., Davari M., Wei W. and Leus R. *Scheduling a single batch machine with non-identical job sizes and incompatible job families* 503

Lin L., Chen Y., Tan K. and Tan Z. *Improved Algorithms for Scheduling with Job Delivery Coordination* 507

Jin H., Liu S. and So R. *Dynamic Incentive Schemes for Managing Dockless Bike-Sharing Systems* 510

Wang H. and Yue L. *Research on electroplating parallel-tank scheduling and optimization of product process selection in PCB manufacturing environment* 530

Sun X. and Li K. *A pickup and delivery problem for the take-out delivery service* 533

Lee J., Park S., An H. and Kang L. *An Optimized Analysis of Crew Scheduling Plan for Railway Operation* 537

Zhou Y. and Guo X. *Multi-objective dynamic programming based on fuzzy optimal selection model and its application to disassembly sequence planning* 541

Chen Z-L., Tang L. and Li F. *Online Supply Chain Scheduling Problems* 546

Chen S-M. and Zeng Q-C. *Online Integrated Optimization Model for Container Rehandling and Pickup Sequence in Container Terminals* 549

Liu P. and Lu X. *Two-machine Openshop scheduling with Two Agents*.....550

He P. and Li K. *An enhanced branch-and-price algorithm for the integrated production scheduling and delivery routing problem* 553

Li Y., Lu L. and Zhang L. *Two single-machine scheduling problems with rejection* 554

Xie J., Gao L. and Li X. *A Modified Teaching-Learning-Based Optimization Algorithm for Distributed Job Shop Scheduling Problem with fuzzy processing time* 555

Li Y., Li X. and Gao L. *A Discrete Artificial Bee Colony for Heterogeneous Distributed Hybrid Flowshop Rescheduling Problem* 556

Wang G., Li X., Gao L. and Li P. *Energy-efficient distributed flow shop scheduling using a modified MOEA/D*.....557

Chau V., Feng S., Li M., Wang Y., Zhang G. and Zhang Y. *How to schedule jobs with limited number of Calibrations* 558

Sun D. and Tang L. *Combinatorial Benders approach to solve the Quay Crane Scheduling Problem* 561

Su L., Huang Y. and Tang L. *Scheduling for refinery processes with consideration of uncertain demand*.....563

Gong H. and Tang L. *Coordinated scheduling of transportation and production on single batch machine based on reinforcement leaning*.....564

Shi F. and Tang L. *Integration Transportation Problem in the Open-Pit Mine via Decomposition based Reinforcement Learning* 565

Yang A. and Zeng Q. Y. *Feasibility of Digital Twin on Digital Port Designing*.....566

Luo D. and Guan Z. *Big data analysis based robotic cellular manufacturing optimization model: Integrated production planning and preventive maintenance scheduling*.....567

Shan L. and Qingcheng Z. *Operation scheduling of automated container terminals* 570

Wang G., Yin Z. and Tang L. *Solving the iron ladles allocation problem in molten iron transportation*.....571

Fan G-Q., Wang J-Q. and Liu Z. *Two-agent scheduling on mixed batch machines to minimize the weighted sum of the agents' makespans*.....572

Fang W., Guan Z. and Yue L. *Distribution route planning for station requirements material with split deliveries in assembly line*.....573

Zhang Y., Bai R. and Qu R. *A Deep Reinforcement Learning Based Hyper-heuristic for Container Terminal Truck Routing Problem under Uncertain Service Times* 576

Li F., Yang D. and Wang S. *A branch-and-price algorithm for bunkering tanker* 580

Liu S. and Pan W. *An integrated machine-learning and optimization approach for a vehicle dispatching and routing problem in logistics industry* 583

Fränti P. and Nenonen H. *Modifying Kruskal algorithm to solve open loop TSP*.....584

Braune R. *Classification-based algorithm selection for a multiprocessor scheduling problem* 591

Christensen S. K. *An Adaptive Large Neighbourhood Search Algorithm for Shift Assignment in Shift-Based Businesses* 595

Chen X., Bai R. and Dong H. *A Multi-Layer GP Hyper-heuristic for Real-Time Truck Dispatching at a Marine Container Terminal* 598

Cailloux J. and Moench L. *Scheduling Jobs in Flexible Flow Shops with Batching and Time Constraints* 603

Luo H. and Yang X. *New bulky goods transportation mode in urban logistics based on Physical Internet* 608

Huo Y. and Zhao H. *Revisit Heuristics for Flowshop Scheduling with Availability Constraint* 609

Xu Z., Lai X. and Liu L. *Coordinating Resource Sharing for Joint Optimization among Carrier Alliances in Liner Shipping* 612

Luo Z. *A branch-and-price algorithm for the electric vehicle routing problem with nonlinear charging function* 613

Xia J. *Routing Optimization with Generalized Consistency Requirements* 614

Plenary Presentations

Tan Kay Chen

Advances in Evolutionary Multitasking for Vehicle Routing

Abstract

Evolutionary multi-tasking (EMT) is a recently emerged search paradigm in the realm of evolutionary computation. In contrast to traditional evolutionary algorithm (EA) which solves a single task in a single run, the EMT conducts evolutionary search concurrently on multiple search spaces corresponding to different tasks or optimization problems, each possessing a unique function landscape. By leveraging the useful knowledge across tasks while the evolutionary search progresses online, the EMT has demonstrated superior optimization performance in terms of search efficiency and effectiveness on continuous as well as discrete optimization problems, when compared to its single-task counterparts. Vehicle routing problem (VRP) is a generalization of the Traveling Salesman Problem (TSP), which has been proved to be NP-hard. To solve the VRP, traditional single-task optimization methods ignored the similarity shared among VRPs, which could lead to enhanced vehicle routing process when properly harnessed. Taking this cue, several EMT algorithms have been proposed towards an enhanced vehicle routing process in the literature. In this talk, I will present our recent progresses on advanced EMT algorithm designs for VRPs, such as Capacitated VRP and VRP with heterogeneous capacity, time window as well as occasional drivers. Some key issues of EMT algorithm designs for VRPs will also be discussed.

Nicholas G. Hall

A New Design for Tournaments

Abstract

The tournament designs that are used for numerous professional and amateur sports events all over the world suffer from a number of well documented deficiencies. These deficiencies typically reduce the fairness, excitement value, profitability, and credibility of those tournaments. We describe a new design that substantially reduces these problems. We validate our design using both data from professional sports tournaments and an extensive sensitivity analysis with simulated data.

José F. Oliveira

Tackling uncertainty in combinatorial optimization problems: using metaheuristics to efficiently co-generate scenarios and solutions

Abstract

Uncertainty is receiving increasing attention, in the past years, from the Operational Research community. Methods that acknowledge uncertainty and incompleteness of information are an important research trend. Scenarios arise as key components in many of these methods, as instruments to deal with uncertainty. However, the scenario generation process is often unrealistically simplified. We propose that metaheuristics, namely based on genetic algorithms, can generate relevant and complex scenarios, without requiring a priori probability distributions. This is of particular interest in practical applications where there are many uncertain parameters, and it is significantly difficult to define their characteristics accurately. To address two-stage stochastic problems, we propose a method based on a co-evolutionary metaheuristic, where solutions and scenarios are generated and evolve in parallel. The goal of the evolution of the solution population is to obtain values for the first-stage decisions that perform well when compared with the scenario population. The goal of the evolution of the scenario population is to diversify the impact of its elements on the value of solutions. This methodology is able to support decision-makers with different risk profiles. To illustrate the method, we apply it to the integrated problem of fleet management and pricing for car rental companies under demand and competitor pricing uncertainty. When planning a selling season, a car rental company must decide on the number and type of vehicles in the fleet to meet demand. The demand for the rental products is uncertain and highly price-sensitive, and thus capacity and pricing decisions are interconnected. Moreover, since the products are rentals, capacity "returns". This creates a link between capacity with fleet deployment and other tools that allow the company to meet demand, such as upgrades, transferring vehicles between locations or temporarily leasing additional vehicles. An ongoing extension of this work is an innovative scenario generator, based on this idea of impact diversity, that quickly provides representative sets of scenarios that can be used to feed not only a genetic algorithm but any stochastic solution method.

Lixin Tang

Data Analytics and Optimization for Steel Industry

Abstract
Some interesting topics on the production, logistics and energy analytics and optimization in steel industry will be discussed in this talk, including: 1) production scheduling in steel-making and hot/cold rolling operations; 2) logistics scheduling in storage/stowage, shuffling, transportation and (un)loading operations; 3) energy optimization including energy allocation and coordinated planning and scheduling of production and energy; 4) data analytics including dynamic analytics for BOF steelmaking process based on multi-stage modeling; temperature prediction for blast furnace; temperature prediction for molten iron in transportation process; analytics for description, diagnostics, and prediction of energy generation and consumption; temperature prediction for reheat furnace based on mechanism and data; strip quality analytics for continuous annealing based on multi-objective ensemble learning; process monitoring and diagnosis for continuous annealing based on mechanism and data.

Ling Wang

Data-Driven Intelligent Optimization Scheduling

Abstract

Optimization and scheduling problems are the significant issues faced by the manufacturing industry. Over the past few decades, an impressive number of data-driven intelligence algorithms were reported for solving the engineering optimization and scheduling problems (EOSPs). This talk first shows the complexities of the EOSPs; and then introduces a unified framework for the population-base intelligent optimization techniques from a systematic perspective as well as an integrated intelligent optimization framework; finally presents some typical research work in terms of theoretical analysis, constrained optimization and intelligent scheduling algorithms. The primary aim of this talk is to show that intelligent algorithms are powerful and general solution tools for solving the EOSPs, while it is more important to incorporate the problem-specific knowledge into the algorithms for solving specific problems.

Papers

MISTA 2019

Learning Regular Expressions for Interpretable Medical Text Classification Using a Pool-based Simulated Annealing and Word-vector Models

Chaofan Tu • Ruibin Bai • Zheng Lu • Uwe Aickelin • Peiming Ge • Jianshuang Zhao

Abstract We considered a rule-based engine consisting of auto-generated regular expressions for high quality and fully interpretable medical data processing. Existing Deep Neural Network (DNN) methods often present high quality performance in most Natural Language Processing (NLP) applications. However, these methods are often served as “black-box” tools and resulting solutions lack sufficient interpretability, which is a critical requirement for automated medical data processing. There have been some efforts in combining these DNN approaches with rule-based methods (e.g. regular expressions) to improve their accuracy and interpretability but construction of these regular expressions can be extremely labour intensive for large datasets. This research aims to reduce those manual efforts while maintaining high quality medical data processing and interpretability. A modified Simulated Annealing method, called pool-based simulated annealing with word vector model, or PSAW, is proposed to automatically derive regular expressions to form a rule-based classification engine. The

proposed method was tested for 30 medical text classification tasks, each containing half a million real-life data from one of China's largest online medical platforms, the proposed PSAW method showed promising potentials compared to domain experts. Additional results show that the performance of our method can be further improved by combining it with the state-of-the-art DNN approaches. We believe the proposed method can be a vital complementary tool for text classification applications which require high levels of interpretability of the solutions.

1 Introduction

With the prevalence of modern computerised technologies, the chances to boost the accuracy of the auxiliary diagnosis models are manyfold which in turn enhances the doctor/hospital operational efficiency. By using latest technologies in speech recognition, machine vision, Natural Language Processing (NLP), machine learning, and others, data mining is becoming a necessity in the field of intelligent healthcare sector where a huge amount of digital data is available. For instance, the IBM Watson for ontology has demonstrated concordance rates of 96% for lung, 81% for colon, and 93% for rectal cancer cases with a multi-disciplinary tumour board in India [1]. The IBM Watson collects the data from 8500 hospitals, insurers, and government agencies [2]. Another popular application of intelligent healthcare is the DeepMind's Streams medical diagnosis mobile application. The application sends alerts to nurses and doctors when a patient's readings look abnormal through analysing the medical records of 1.6 million patients [3]. The availability of vast quantity of digitised healthcare and patient data plays an important role in the auxiliary diagnosis models.

This work is based on the collaboration with a very large online medical platform that allows patients to see a doctor online. The average number of daily consultation requests submitted to this platform in 2017 exceeded 370,000 [4]. In order to assign appropriate doctors across the different disciplines such as gynaecology, paediatrics, dermatology and so on, the system has to first deal with the classification of medical inquiries, which are sentence-level patients' complaints in text format. Any mistakes in the classification process will lead to doctor miss-assignment and therefore reduce the system overall efficiency, especially in this real-time online scenarios. For the sake of not only reducing the number of employees who should be available 24/7 handling the medical reception but also enhancing the platform's operational efficiency, it is very important to implement an automated classification system.

Some state-of-the-art approaches such as DNN have provided good performance in various NLP tasks while no human effort is required once a suitable model has been built, but their main drawback is the lack of interpretability for medical use in such "black box" approaches. Additionally, it may even require complete retraining when one particular classification task is updated. Therefore, the medical industry prefers to ensure good interpretability of system for rigorous validation requirements even if additional professional human resources are required.

It is impossible to deal with massive classification tasks by manual confirmation. Regular expressions are widely used in text-matching techniques that are fully interpretable compared to learning-based models. The regular expression based classification engine constructed by domain experts can be used to verify the most possible results from DNN model, meanwhile not only enhancing the interpretability and improving the precision of entire classification system. The rules engine plays an important role especially for classification tasks that are particularly challenging for DNN. Its main advantage is its shortcomings: manual construction of the rules require domain knowledge and a considerable amount of human work.

In order to achieve a good trade-off between interpretability and human efforts for the medical text classification problem, this paper proposes a fully automated system for learning regular expressions to capture relevant words and synonyms, discard irrelevant words and form good word sequences for good readability and further revision. The contributions of this paper are as follows:

- A specially designed structure of solutions is proposed to reduce the complexity whilst maintaining flexibility;
- A method called PSAW is proposed, combining pool-based Simulated Annealing and word-vector model to enhance the readability of auto-generated regular expressions.
- Impacts of parallel and iterative strategies for tasks of learning regular expressions have been intensively explored by comparing two extended versions of PSAW.
- On real industry datasets, the performances of PSAW and domain experts were compared.

2 Related works

Text classification involves assigning a text document to a set of pre-defined classes automatically. Classification is usually done by using significant words or features extracted from the raw textual document. This is a typical supervised machine learning task with predefined classes and training text documents with class labels [5]. Automated text classification usually includes steps such as pre-processing (eliminating stop-words, etc.), feature selection using various statistical or semantic approaches, and text modelling [5]. Until late 80's, text classification task was based on Knowledge Engineering (KE), where a set of rules were defined manually to encode the expert knowledge on how to classify the documents given the categories [6]. Since there is a requirement of human intervention in knowledge engineering, researchers in 90's have proposed many machine learning techniques to automatically manage and organise the textual documents [6]. The accuracy of those machine learning techniques is comparable to that of human experts and no artificial labour work from either knowledge engineers or domain experts is needed for the construction of a document management tool [7].

Text classification involves challenges and difficulties. First, it is difficult to capture high-level semantics and abstract concepts of natural languages. Second, semantic analysis, a major step in designing an information retrieval system, is not easy to be well understood by machines. Third, the high dimensionality (e.g. thousands of feature vectors) of data poses negative influences for classification tasks [8].

In order to perform text classification, we first need to use a propper text representation. Bag of Words (BoWs) is one of the most commonly used methods to represent a document. By using a fixed global vocabulary, the BoWs use a vector to represent a text document based on the frequency count of each term in the document. This kind of methods of text representation is called Vector Space Model (VSM) [9]. Unfortunately, BoWs/VSM representation scheme has its own limitations such as very high dimensionality of the representation, loss of correlation with adjacent words, and absence of semantic relationship [10]. Another VSM-based method is a neural network model called Word2vec which is used in this paper for extracting word embeddings, which was proposed by Mikolov et al. in 2013 [11, 12]. This kind of fixed-length vector representation (often hundreds of dimensions) trained by deep learning model has shown the ability to carry semantic meanings. This technique can be used in various NLP tasks such as text classification, speech recognition, and image caption generation [13].

After text presentation, word embeddings or numerical representations for text feature extraction can be fed into plain classifiers like the Naïve Bayes, decision tree, neural network, support vector machine, hybrid approaches etc. [8]. The Naïve Bayes classifier is the simplest probabilistic classifier used to classify the text documents into predefined labels [8]. The K-Nearest Neighbour classifier is a non-parametric method which can be shown that for large datasets the error rate of the 1-Nearest Neighbour classifier is not likely to be larger than twice the optimal error rate [8]. Centroid based classifier is a popular supervised approach used to classify texts into a set of predefined classes with relatively low computation [8]. Decision trees are the most widely used inductive learning methods [8]. Decision trees' robustness to noisy data and their capability to learn disjunctive expressions seem suitable for document classification [8]. Support Vector Machine (SVM) is a supervised classification algorithm that has been extensively and successfully used for text classification tasks [8]. Neural network

based text classifiers are also prevalent in the literature, where the input units are denoted as feature terms, the output unit(s) is the category or categories of interest, and the weights on the edges connecting units form dependence relations [8].

With a simple convolutional neural network (CNN) built on top of word-vector models, A series of experiments of sentence-level text classification problems suggest that unsupervised pre-training of word vectors is an important ingredient in deep learning for NLP [14]. Neural network based approaches are strong in terms of precision and recall but usually less interpretable because those "black box" models cannot be logically explained [15]. In addition, those "black box" approaches cannot be quickly modified except retraining the whole neural network models [16]. To address those difficult issues, some related work has shown that regular expressions can be effectively used to solve text classification problems in an interpretable way.

A novel regular expression discovery (RED) algorithm and two text classifiers based on RED were designed to automate both the creation and utilisation of regular expressions in text classification [17]. The proposed RED+ALIGN method correctly classifies many instances that were misclassified by an SVM classifier. A novel transformation-based algorithm, called ReLIE, was developed to learn such complex character-level regular expressions for entity extraction tasks and related experiments demonstrated that it is effective for certain classes of entity extraction in text documents [18].

Automated regular expression generation can also be viewed as a data-driven optimisation problem. In this paper, a well-known simulated annealing hyper-heuristic [19] has been adapted for learning regular expression based classifiers for text classification. The choice of this approach is based on the fact that there are naturally multiple neighbourhood operators available for generating regular expression variants and hyper-heuristics can learn to orchestrate the selections of different operators to achieve high performance across different problems. It has been shown that specially designed neighbourhood operators of SA will lead to better performance [15].

3 Problem description

Formally the problem can be defined as follows: given a set of predefined classes C (or medical templates in the context of our application) and a set of text inquires Q, the problem is to classify each inquiry $q \in Q$ to one of the classes $c \in C$ automatically based on a set of previously labelled examples by medical experts. Table I shows examples of the classification, text inquiry is usually a piece of text information given by the user, describing the medical conditions or problems; the classification task is to select the most appropriate medical template for this inquiry.

TABLE I. EXAMPLES OF MEDICAL TEXT CLASSIFICATION

Text inquiry	Medical template
"My girl is three years old and always coughs without fever, what can I do for her?"	*Cough: 1-3 years old child*
"I have been suffered from pain in my lower abdomen for 3 weeks."	*Adult bellyache*
"The acne grows on the back, and recently it is a little itchy near it."	*Folliculitis*
"I have a serious hair loss, how to make hair?"	*Hair loss*

Let R be a regular expression designed for the classification of class (or medical template in our application) C (denote $|C|$ be the number of inquiries in class C), and let $M(R, Q) \in Q$ be the set of all medical texts matched by R in Q; Denote $M_p(R, Q) = \{q \in M(R, Q): q \text{ is an instance of } C\}$ to the set of all correctly matched entries (medical text inquires) and denote $M_n(R, Q) = \{q \in M(R, Q): q \text{ is not an instance of } C\}$ is the set of all mismatched entries (medical

text inquires). Like many classification problems, this problem also has two performance indicators, which are precision and recall as bellows:

$$precsion(R,Q) = \frac{|M_p(R,Q)|}{|M_p(R,Q)|+|M_n(R,Q)|} \quad (1)$$

$$recall(R,Q) = \frac{|M_p(R,Q)|}{|C|} \quad (2)$$

The well-known F-measure (also called F-score) can be a better single metric when compared to precision and recall. Given a non-negative real β for users' preference, it can be expressed as:

$$F_\beta(R,Q) = \frac{(1+\beta^2)\times precsion(R,Q)\times recall(R,Q)}{\beta^2 \times precison(R,Q)\times recall(R,Q)} \quad (3)$$

The problem of automated learning of regular expression based classifiers for medical text in this paper can be formally expressed as an optimization problem for regular expression *R*. Let *S* be the solution space of *R*, for a given class of *C* and labelled dataset *W* which can be divided into a positive part and a negative part, the problem is to find a solution with the optimal objective function F_β from the solution space *S*. So this problem can be defined as:

$$R_{target} = argmax_{R\in S} F_\beta(R,Q) \quad (4)$$

4 Methodology

4.1 Proposed Structure

In this problem, each solution is encoded as a vector of *m* regular expressions < R_1, R_2, ..., R_m >. To check whether a particular inquiry belongs to a class (or template), the regular expressions in the vector is executed one by one sequentially in the same order of the vector for the inquiry under consideration. If the inquiry is matched by any of regular expressions, the inquiry is said to be in the class, otherwise, it is not in the given class. Each regular expression R_i is derived via a combination of functions and terminals defined in Table I and follows a global structure of two parts P_i and N_i concatenated by the *NOT* function #_#, where P_i tries to match all positive inquiries and N_i is then used to filter out the list of falsely matched inquiries by P_i. That is, each regular expression has the following format:

$$R_i = (P_i).(\#_\#(N_i)) \quad (5)$$

TABLE II. FUNCTIONS AND TERMINALS

Name	Label	Description
NOT	*#_#*	Function to negate a given expression.
Sub-expression	*e*	An expression or term obtained through a combination of words and functions listed below.
OR	|	Function to test logic or of two expressions.
Words	*w*	List of keywords extracted from the target text set.
AND	.	Function to test logic AND of two expressions.
Adjacency	*{a, b}*	Function to test whether the distance between two words w_1 and w_2 are in the range [a, b], which is an extended function of the AND function.

For the purpose of achieving better readability of regular expressions R_i and limiting the size of search space, the following constraints are applied to each of regular expressions R_i :

1) Each regular expression R_i has at most one *NOT* function.
2) The positive part P_i or negative part N_i is only composed of *OR* functions and it is defined as the *outer OR structure* as below:

$$R_i = (e_{p1}|e_{p2}|\ldots|e_{pm}).\ (\#_\#(e_{n1}|e_{n2}|\ldots|e_{nn})) \quad (6)$$

3) Function *OR* can be used in any sub-expression *e* and it is defined as the *inner OR structure*, but there should not be any other nested functions except *OR* in *e*. That is, the sub-expression $w_1|(w_2|w_3)$ is acceptable because the nested function is *OR* but the sub-expression $w_1|(w_2 \cdot w_3)$ is not permitted since the nested function is *AND*.

4) Function *AND* used in any sub-expression *e* can contain nested functions of both *AND* and *OR*. For example, both the sub-expression $w_1 \cdot (w_2 \cdot w_3)$ and the sub-expression $w_1 \cdot (w_2|w_3)$ are acceptable.

The *outer OR structure* is used to directly compose positive part P_i or negative part N_i according to condition 2, while the *inner OR structure* used in any sub-expression *e* cannot contain any nested function except *OR* due to condition 3. Through the above restrictions, the overall structures of regular expressions R_i have been limited to a maximum of two levels of nested *OR* structures.

Corollary 1 With the same terminals and functions listed in Table II, there always exists one or more regular expressions that satisfy all the above conditions and is equivalent to any expression without any constraint of structures.

Proof: For condition 1, it's obviously because the *NOT* function is essentially one kind of set operation, multiple *NOT* functions can be reduced to one finally;

For condition 2, it's evident that any other single or multiple functions can apply *OR* function to the outer layer of itself, because the *OR* function has a lower priority than any other function except *NOT*;

For condition 3 and 4, if a sub-expression *e* of P_i is directly composed of *OR* function and the *OR* function contains an *AND* function, P_i can be clearly transformed into a new expression which meets the condition 2 as below:

$$\text{Let } e_{p2} = w_1|(w_2 \cdot w_3)\ ,$$
$$P_i = (e_{p1}|(w_1|w_2 \cdot w_3)|\ldots|e_{pm}) = (e_{p1}|w_1|w_2 \cdot w_3|\ldots|e_{pm}) \quad (7)$$

If a sub-expression *e* of P_i is composed of one *AND* function and the *inner OR structure* contains an *AND* function such as $(w_1|(w_2 \cdot w_3)) \cdot w4$, P_i can be also transformed into a new expression which meets the condition 2 and 4 as below:

$$\text{Let } e_{p2} = (w_1|(w_2 \cdot w_3)) \cdot w_4\ ,$$
$$P_i = (e_{p1}|((w_1|(w_2 \cdot w_3)) \cdot w_4)|\ldots|e_{pm}) = (e_{p1}|w_1 \cdot w_4|(w_2 \cdot w_3) \cdot w_4|\ldots|e_{pm}) \quad (8)$$

End of proof. That is, although we restrict the possible formats of our regular expressions to one kind of two-layer nested structures, their expressiveness is not reduced. These conditions not only simplify the structure of solutions but also contribute to enhancing the readability and interpretability.

4.2 Solution Pool Mechanism

According to problem descriptions and structures defined above, the medical text classification problem in this paper is transformed into a combinatorial optimisation problem. The Simulated Annealing algorithm is a large-scale combined problem global optimisation algorithm, which is widely used to solve the NP-hard combination optimisation problem.

An improved Simulated Annealing method is applied as the evolutionary computation algorithm to solve the problem in this paper, with a well-designed solution pool mechanism to enhance the diversity of solutions as shown in figure 1:

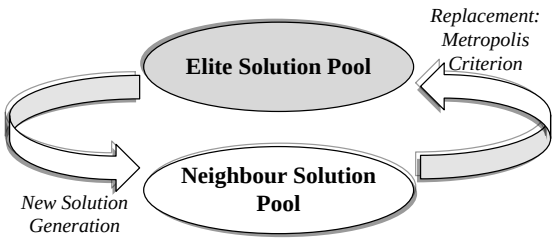


Fig. 1. Solution pool mechanism

The number of solutions in the elite solution pool is set to a fixed value, and the same amount of new solutions are transformed from an initial solution for the initialization of the elite solution pool.

The number of solutions in the neighbour solution pool is the same as the elite solution pool. Each solution in the elite solution pool will produce a new solution in each iteration during the entire period, then a totally updated neighbour solution pool can be formed from those all newly-generated solutions.

The elite solution pool will be also updated in each iteration during the entire evolution. For each solution in the updated neighbour solution pool, one solution in the elite solution pool will be randomly selected for comparison and replacement. The used acceptance criterion for replacement adopts Metropolis criterion based on Simulated Annealing algorithm.

Every time the best solution in the elite solution pool is always retained during the whole period. The details of the proposed solution pool mechanism are shown below in figure 2.

Set the capacity of a solution pool to be N_{pool};
Define a set of each elite solution S_{e_i} (i = 1,…, N_{pool}) as the elite solution pool P_e;
Define a set of each neighbour solution S_{n_j} (j = 1,…, N_{pool}) as the neighbour solution pool P_n;
Set the best solution in P_e as S_{e_best};
Solution Replacement:
begin
  j = 1
  while ($j < N_{pool}$) **do**
    select S_{n_j};
    select S_{e_i} randomly from P_e;
    let δ be the difference in the evaluation function between S_{e_i} *and* S_{n_j};
    if the Metropolis criterion of simulated annealing is satisfied by δ
      $S_{e_i} = S_{n_j}$;
    endif
    j ++;
  add S_{e_best} into P_e temporarily;
  sort P_e by the evaluation function of each solution S_{e_i} (i = 1,…, N_{pool} +1) ;
  P_e = { S_{e_i} | i = 1,…, N_{pool} }
  $S_{e_best} = S_{e_1}$
end

Fig. 2. Pseudo-code of the proposed solution pool mechanism

4.3 Initialisation

From the above description, an existing initial solution is a precondition for the initialisation. In order to balance speed and readability, we carry out a method using word frequency and similarity comparison to quickly generate a group of keywords as an initial solution. The specific steps are described below:

1) First, if the frequency of a word in the positive dataset exceeds the predetermined times defined as TD_F, this word will be added into the set of keywords;

2) Second is sorting the keyword set by word frequencies and calculating cosine similarities between any two different words' vectors in the top N_W number of keywords. If the cosine similarity exceeds the predetermined value defined as TD_S, these two words are considered as a group of same subject words;
3) Third, one group of subject words is randomly selected and those words are connected as an *inner OR structure* as an initial solution with an empty negative part. Below is an example:

((headache | dizzy | giddy | dizziness)).(#_#())

One or more regular expressions generated by the above steps from the initial solution S_{init}.

4.4 Neighbourhood Operators

In our proposed method, it is decided by a random strategy firstly whether to update the positive or negative part of solutions, then seven specially designed neighbourhood operators are used for generation of a new solution.

O1: **Adding *OR* type 1** is an operator to add a word to the *inner OR structure*. First randomly select ten words from the set of positive words or the set of negative words. Then randomly select an existed word from the inner nested *OR* structure, and calculate the cosine similarity Sim_i (i = 1,…, 10) between the existed word and the other ten words based on pre-training word vectors. Finally, only one word of the ten will be selected to add to the *inner OR structure* with the largest probability $Prob_i$ as below:

$$Prob_i = \frac{Sim_i}{\sum_1^{10} Sim_i} \tag{9}$$

The extension of the inner *OR* structure by this type of function combines the information of the pre-trained deep learning model, word2vec, so that the readability of regular expression has been considered during the evolutionary.

O2: **Adding *OR* type 2** is an operator to add a sub-expression to the outer *OR* structure. First, randomly select a word from the positive or negative word set. If the selected word does not exist in the outer *OR* structure, add this word into it; if the word is already in the outer *OR* structure, then randomly select another word to form a non-repeating *AND* (or *Adjacency*) sub-expression *e* and finally add *e* into the outer *OR* structure.

O3: **Removing *OR*** is an operator to randomly delete a sub-expression that makes up the outer or inner *OR* structure as inverse operations of O1 and O2.

O4: **Adding *AND*** is an operator to extend the *AND* (or *Adjacency*) structure in the sub-expression in the outer *OR* structure. Randomly pick a word to insert into an existing *AND* (or *Adjacency*) structure or form a new *AND* structure with an existing word.

O5: **Swap** is an operator to exchange the positions of any two sub-expressions or words in the *AND* (or *Adjacency*) structure.

O6: **Distance** is an operator to randomly change the maximally permitted distances between two expressions or words based on a given *Distance Table*. So the *AND* function can be considered as an *Adjacency* structure with unrestricted distance.

O7: **Removing *AND*** is an operator to randomly delete one sub-expression *e* or word *w* that makes up one *AND* (or *Adjacency*) structure, as an inverse operator of O4.

4.5 Solution Decoding and Evaluation

Each regular expression R_i in solution pools should be decoded to a valid regular expression that can be passed through the general regular expression matching engine. There are two main points to note here. The logical symbols defined in this paper are not exactly the same as the symbolic system of regular expressions; The *NOT* function defined in our system does not exist in regular expression matching engine, so the positive and negative parts of R_i need to be separately handled. An example of this conversion should be done as follows:

The positive part of the converted R_i:

$$.*(((w_1|w_2|w_3).*w_4|w_5.\{0,10\}(w_6|w_7|w_8)|w_9|w_{10}|\ldots|w_p)).* ;$$

The negative part of the converted R_i :

$$.*(((w_1|w_2|w_3|w_4|(w_5|w_6).\{0,10\}w_7|w_8|\ldots|w_n)).*$$

The performance of each regular expression based solution or classifier will be evaluated based on the F-measure value according to the above description in section 2. In this paper, the parameter β for F-measure is set to 0.2 for the purpose of giving more attention to precision.

4.6 Overall Algorithm

In this paper, a modified Simulated Annealing method, called pool-based simulated annealing with word vector model, or PSAW, is proposed to automatically derive regular expressions to form a rule-based classification engine. Below figure is the overall flow of PSAW.

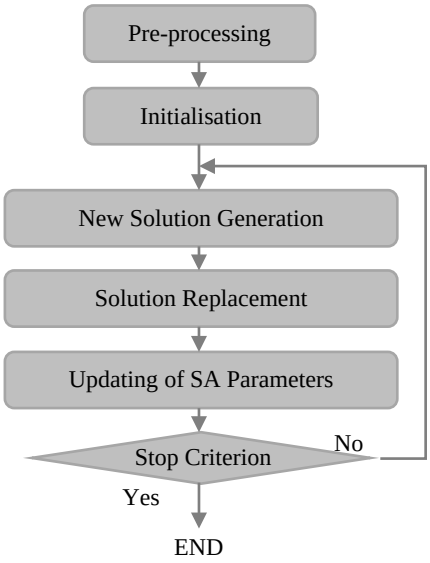


Fig. 3. The overall process of PSAW

Before initialisation the pre-processing includes dividing the training data set into positive and negative sets, performing Chinese word segmentation, removing stop words, and pre-training the Word2vec model.

The initial solution has been generated after the initialisation. In the beginning, the elite solution pool will be fully filled with new solutions transformed by the initial solution.

According to the Metropolis criterion, solutions in the elite solution pool may be replaced by solutions in the neighbour solution pool. Parameters such as the temperature of classic simulated annealing would be updated in each iteration. The program terminates when the total iterations are finished or the stop condition is met.

4.7 Iterative and Parallel Strategies

To further explore the impacts of different running strategies in computational time and performance, we designed and implemented an extended version called PSAW-I for iterations strategy and another extended version called PSAW-P for parallelism.

1) PSAW-I: It is considered to improve the recall by learning regular expressions iteratively, that is, before learning the next classifier, those text inquiries matched by existing classifiers should be filtered first in the training set.
2) PSAW-P: Consider the method of pre-dividing the positive training dataset for parallel acceleration. When the last parallel task is terminated, all the sub-solutions are merged as one solution for the whole. Pre-dividing is based on semantic clustering on our datasets and also set random division as a comparison baseline.

5 Experiment

5.1 Data and Parameter Settings

Because of the collaboration with a very large online healthcare platform, the experiments in this paper are based on high-quality training and test data from the real production environment. The numbers of real-life medical text inquiries in training set and test set are 1,800,000 and 500,000 respectively. If not stated separately, the following parameters are used in this paper.

1) For the initialization: $TD_F = 5$; $N_W = 100$; $TD_S = 0.75$;
2) For the neighbourhood operators: *Distance Table* = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 100] ;
3) For pool-based simulated annealing: starting temperature $T_S = 0.5$; stopping temperature $T_E = 0.05$; solution pool capacity $N_{pool} = 10$; Total iteration $K = 1000$;
4) For F-measure: $\beta = 0.2$.

5.2 Evaluation of Solution Pool Mechanism

In this experiment, we controlled the variables N_{pool} and K for learning one regular expression for the same template C_1 to evaluate the solution pool mechanism. The N_{pool} of groups 1~3 were set to 1, 10, 50 and the N_{pool} of group 4 was set to 1 to represent the classic simulated annealing without the proposed mechanism. The total number of new solutions in group 4 was set as the same as group 2.

In TABLE III the results of groups 1~3 show that the higher the N_{pool}, the more time cost and the better the performance of F-measure (F_m).

TABLE III. A COMPARISON OF POOL CAPACITIES

Group	N_{pool}	K	F_m	**Time** (min)
1	1	1000	0.60	8
2	10	1000	0.76	59
3	50	1000	0.77	364
4	1	10000	0.72	130

The comparison between group 4 and 2 shows that the solution pool mechanism significantly not only enhances performance and but also reduces time cost. This is because the solution update process of group 4 would make its solution more and more complex to increase the evaluation time and this mechanism in group 2 can improve the diversity of solutions.

5.3 Comparison of Iterative and Parallel Strategies

We have tested the proposed PSAW and its two extended versions of PSAW-I, PSAW-P on six different medical text classes $C_1 \sim C_6$. For further explorations, PSAW-P version adapted two different methods, that are separately clustering division and random division. All solutions were set to contain three regular expression based classifiers. One PSAW-P group was set to use the widely-used k-means clustering method to divide the training dataset into three different

subsets for parallel processing, while another PSAW-P group was set to use the random trisection as a comparison baseline.

In TABLE IV and V, the PSAW-P version with k-means clustering showed the highest level of precision and the least time cost; the PSAW-I version showed the highest level of recall, The original PSAW version presented the most time cost, while its average of F-measure value is the best.

TABLE IV. A COMPARISON OF PSAW AND PSAW-I

Classes	**PSAW**				**PSAW-I**			
	Precision	***Recall***	F_m	***Time**(min)*	***Precision***	***Recall***	F_m	***Time**(min)*
C_1	**0.89**	0.63	**0.87**	266	0.76	**0.76**	0.76	220
C_2	0.69	0.41	**0.68**	335	0.56	**0.50**	0.56	300
C_3	**0.71**	0.11	0.58	417	0.52	**0.25**	0.50	372
C_4	**0.93**	0.33	**0.87**	394	0.81	**0.78**	0.81	333
C_5	0.84	0.69	0.83	405	0.87	**0.83**	0.87	345
C_6	**0.93**	0.61	**0.91**	396	0.84	**0.65**	0.83	239
AVG	0.83	0.46	**0.81**	369	0.73	**0.63**	0.72	302

TABLE V. A COMPARISON OF CLUSTERING AND RANDOM METHODS FOR PSAW-P

Classes	**PSAW-P** (Clustering)				**PSAW-P** (Random)			
	Precision	***Recall***	F_m	***Time**(min)*	***Precision***	***Recall***	F_m	***Time**(min)*
C_1	0.87	0.42	0.83	**86**	0.86	0.52	0.84	110
C_2	**0.72**	0.29	0.68	**119**	0.62	0.26	0.59	125
C_3	**0.71**	0.11	**0.59**	**138**	0.66	0.09	0.54	144
C_4	0.92	0.37	**0.87**	**133**	0.92	0.31	0.86	128
C_5	**0.89**	0.54	**0.87**	**136**	0.86	0.51	0.84	134
C_6	**0.93**	0.48	0.89	**108**	0.92	0.48	0.89	124
AVG	**0.84**	0.37	0.80	**120**	0.81	0.36	0.77	128

5.4 Performance Distribution

The PSAW method has been further applied to learn regular expressions for 30 independent medical templates for evaluating its performances compared to medical experts. The following figure 4 is the precision and recall distributions of auto-generated classifiers by PSAW on the test dataset, compared to those manual regular expression based classifiers written by domain experts.

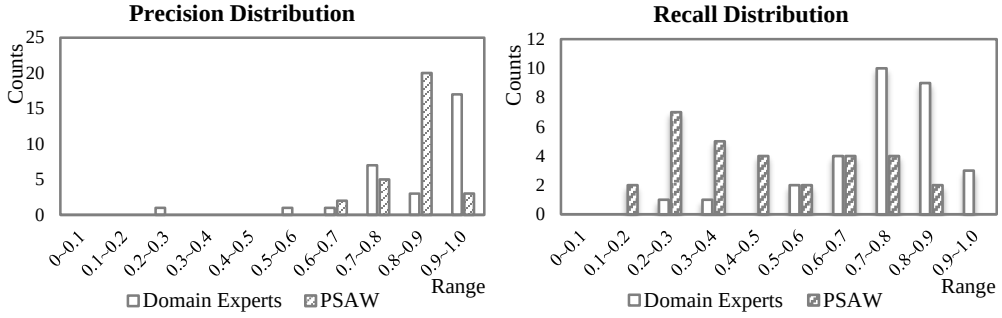


Fig. 4. Precision and Recall Distribution

The precision of most classifiers by domain experts exceeds 0.9 and auto-generated classifiers mostly exceed 0.8. The recall of most classifiers by domain experts exceeds 0.6, while the distribution of auto-generated classifiers is more uniform. The reason may be the parameter for the evaluation function was set to pay more attention to precisions rather than recalls ($\beta = 0.2$).

5.5 Practicality Evaluation

Those auto-generated regular expressions can be used as secondary verification for fully interpretable medical data processing. 50 manual classifiers made by medical experts and 50 auto-generated classifiers by PSAW are randomly selected for third-party practicality blind evaluation using the below score table in TABLE VI.

TABLE VI. SCORE TABLE

Score	Descriptions
1	Cannot be used
2	Can be used after a lot of revisions
3	Can be used after some revisions
4	Can be used after minor revisions
5	Can be used directly without revisions

The score distribution of third-party blind evaluation is shown below in Figure 5.

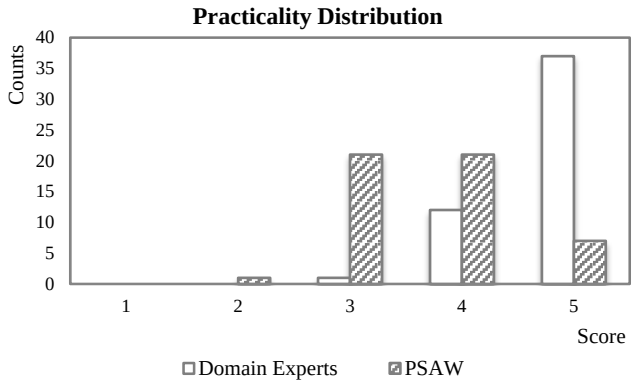


Fig. 5. Third-party blind evaluation

According to the third-party blind evaluation, most auto-generated classifiers by PSAW are well readable and can be applied to practical use after some or minor revisions, which benefits from both the proposed solution structures and the use of word vector model.

5.6 Improvement Evaluation

one of China's largest online medical platforms, the proposed PSAW method showed promising potentials compared to domain experts. Additional results show that the performance of our method can be further improved by combining it with the state-of-the-art DNN approaches. We believe the proposed method can be a vital complementary tool for text classification applications which require high levels of interpretability of the solutions.

Through the auto-generated regular expression based classifiers, a fully interpretable rule-based engine was derived for medical text classification. On the real medical industry dataset with different 140 text classification tasks, the rule-based engine was used to perform secondary verification of the most 5 possible results from a state-of-art DNN model that is being used in the very largest online medical platform of our collaborator. Below is the comparison of the new method of combining the rule-based engine with the DNN and the DNN in Figure 6.

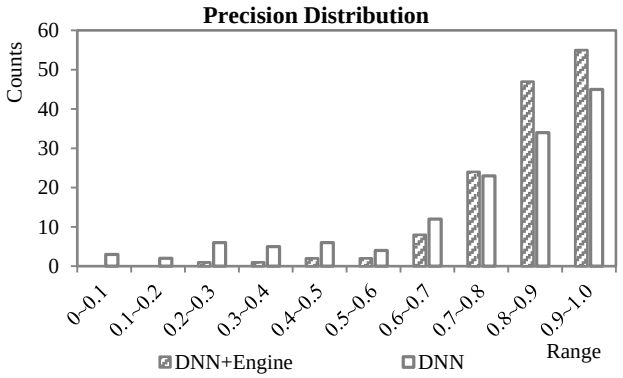


Fig. 6. Improvement Evaluation

The test showed that the combined use of the rule-based engine brought an average 9.35% improvement (from 74.94% to 84.29%) in precision compared to the only use of DNN model meanwhile ensuring good interpretability and full control of results. It can be clearly seen that the new method significantly reduced the number of tasks with low precision of less than 60% and significantly increased the number of tasks with high precision of more than 80%.

6 Conclusion and future research

In this work, construction of regular expressions for medical text classification is transformed into a combinatorial optimisation problem and solved a variant Simulated Annealing method, called, PSAW, which combines the classical simulated annealing with word vector model (pre-trained word2vec model). In addition, iterative and parallel strategies have been explored for further improvement in computational time and performance. Computational results on real-life instances show that PSAW is able to generate promising solutions. Although those auto-generated classifiers by PSAW cannot outperform experts' classifiers on all instances, most of them are fully interpretable and can be served as drafts for domain experts for further improvements. We also integrated the regular expression classifiers with a state-of-art DNN classification model and tested them on real-life data. Considerable improvements were obtained.

Future research directions include using GPU to further speed up the algorithm, adoption of a multi-objective optimization model for regular expression generation, and theoretical analysis for more efficient regular expression encoding in the context of medical text classifications.

Acknowledgements This work is partly supported by the National Natural Science Foundation of China (Grant No. 71471092), Natural Science Foundation of Zhejiang Province (Grant No. LR17G010001), Ningbo Municipal Bureau of Science and Technology (Grant No. 2014A35006, 2017D10034).

References

1. IBM. Bringing confident decision-making to oncology. [cited 2018 24th June]; Available from https://www.ibm.com/watson/health/oncology-and-genomics/oncology/.
2. FOX, B.I. IBM Watson: Artificial Intelligence Meets Healthcare 2017 [cited 2018 24th June]; Available from https://www.asapnet.org/files/June2017/Presentations/ASAPJune17_Presentations07_Fox.pdf.
3. Ram, A. DeepMind develops AI to diagnose eye diseases. 2018 [cited 2018 24th June]; Available from: https://www.ft.com/content/84fcc16c-0787-11e8-9650-9c0ad2d7c5b5.

4. Limited, P.A.H.a.T.C. Introduction of Company. 2018 [cited 2018 24th June]; Available from: http://www.pahtg.com/media/1168/pagd-presentation_-cn-website-version-final-1.pdf.
5. Dalal, M.K. and M.A. Zaveri, Automatic text classification: a technical review. International Journal of Computer Applications, 2011. 28(2): p. 37-40.
6. Mitra, V., C.-J. Wang, and S. Banerjee, Text classification: A least square support vector machine approach. Applied Soft Computing, 2007. 7(3): p. 908-914.
7. Fung, G. P. C., Yu, J. X., Lu, H., & Yu, P. S., Text classification without negative examples revisit. IEEE transactions on Knowledge and Data Engineering, 2006. 18(1): p. 6-20.
8. Harish, B.S., D.S. Guru, and S. Manjunath, Representation and classification of text documents: A brief review. IJCA, Special Issue on RTIPPR (2), 2010: p. 110-119.
9. Salton, G., A. Wong, and C.-S. Yang, A vector space model for automatic indexing. Communications of the ACM, 1975. 18(11): p. 613-620.
10. Bernotas, M., Karklius, K., Laurutis, R., & Slotkienė, A., The peculiarities of the text document representation, using ontology and tagging-based clustering technique. Information Technology and Control, 2015. 36(2).
11. Mikolov, T., Chen, K., Corrado, G., & Dean, J., Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J., Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems 2013:p. 3111-3119.
13. Rong, X., word2vec parameter learning explained. arXiv preprint arXiv:1411.2738, 2014.
14. Kim, Y., Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882, 2014.
15. Moreo, A., Eisman, E. M., Castro, J. L., & Zurita, J. M., Learning regular expressions to template-based FAQ retrieval systems. Knowledge-Based Systems, 2013. 53: p. 108-128.
16. Dalianis, H., J. Sjöbergh, and E. Sneiders. Comparing manual text patterns and machine learning for classification of e-mails for automatic answering by a government agency. in International Conference on Intelligent Text Processing and Computational Linguistics. 2011. Springer. 2011:p.234-243.
17. Bui, D.D.A. and Q. Zeng-Treitler, Learning regular expressions for clinical text classification. Journal of the American Medical Informatics Association, 2014. 21(5): p. 850-857.
18. Li, Y., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., & Jagadish, H. V., Regular expression learning for information extraction[C]//Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2008: p. 21-30.
19. Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., & McCollum, B., A simulated annealing hyper-heuristic methodology for flexible decision support. 4OR, 2012. 10(1): p. 43-66.

MISTA 2019

Satisfying Allocation of Plots to Tenants

Lior Aronshtam · Hagai Ilani · Elad Shufan

Abstract We present a real-life problem of assigning plots to tenants in a new neighbourhood in Israel. The problem deals with clients who compete for resources. In the case described, the clients are families and the resources are the plots allocated for house construction. The presented problem is equivalent to a bipartite hypergraph matching problem, which is known to be NP-hard. We present the problem characteristics, the proposed solution, and the results. In particular, we offer a flexible objective function, which enables the finding of a solution that is adjusted to the community needs. Our motivation comes from issues of client expectations and satisfaction. We pay special attention to determination of the number of preferences that each competitor should choose, so that she will likely receive one of these choices.

Keywords Applications · Real World Scheduling

1 Introduction

The work we present was inspired by a real problem of matching land plots to families in Israel. This is a common problem, both in neighbourhoods within cities and in the periphery; one that exists all over the world. The problem consists of m families and n ($\geq m$) plots. Each family should be assigned one plot. Usually, each family has focused its interest on several plots that are

Lior Aronshtam
Department of Software Engineering, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: liorar1@sce.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Elad Shufan
Physics Department, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: elads@sce.ac.il

mostly preferred by the family, as well as other plots that are less favored. In Israel, the standard method of assigning plots to families in such projects consists of two stages. At the first stage, a random order of the families is uniformly drawn. At the second stage, each family chooses, in its turn, a plot from the set of the remaining available plots, i.e., a plot that had not previously been selected by a family with an earlier choice position. In other words, the first lottery family chooses a plot from any of the n plots, the second family chooses a plot from the remaining $n-1$ plots, and so on. The method is very beneficial for the families that are ordered first, and gets less beneficial for families ordered towards the end of the list. The last to choose selects from a small number of plots, or simply gets what is left. This usually allocates less favoured plots for end-of-the-list families. We offer a different preference-based procedure that we believe increases the allocation fairness and the overall customer satisfaction. The problem, which has been known in the scheduling community for many years, is of matching objects to agents [8, 1,5,4]. Because families usually have different needs and desires, it is generally possible to arrive at a better overall assignment, in which more families will come out satisfied. Every family gives the organizers a preference-ordered list of size l. The aim is to algorithmically find the best possible assignment.

Theoretically speaking, the organizers can ask the families to order all the plots by their preferences. Indeed, papers dealing with this type of allocation problem usually assume a complete preference list [5]. In our procedure, the families are asked to submit a partial preference list of a given size l. This is done for two main reasons. The first is to create realistic expectations for the families. Namely, the organizers declare that there is a high probability for the procedure to allocate each family with one of their l preferable plots. The second relies on the tendency of people creating a large preference list to be less meticulous about their preferences reaching some point in the list of their preferences. Suppose, for example, a setting in which the families choose a complete list, and one given family expects to receive one of the first three options. This family might rank only the first three options with care. But, it may be that there is a high probability of any of the families getting one of their first five preferences. By setting $l = 5$, the organisers advise the families to rank with care all their first five preferences. Therefore, one of the new attributes in our paper is establishing the value of l. Later in the paper we discuss the process we used to establish this value.

As mentioned, the aim of the procedure is to give an assignment that most satisfies the families in the community. However, the notion of a "most satisfies" assignment is not clear cut; there is a range of possibilities. Obviously, a good assignment must be Pareto optimal, i.e., there should not be any other assignment that improves the assigned plot for one family, which does not make things worse for any other family. However, usually there can be more than one Pareto optimal assignment. The best Pareto optimal assignment depends on the perception of the members of the community. There are two standard attitudes for setting the objective function. Roughly speaking, the first is to maximize the assignments of tenants to their most preferred plots [9,10,12].

The second, is to minimise the assignments of tenants to their less preferred plots [7].

The paper offers a mathematical method in which the adjustment of a set of parameters can shift from one objective function to the other, proceeding through a range of intermediate, more moderate, options. Thus, the paper offers the organizers a method to choose an objective function such that the outcome of the procedure will fit their desires.

Because some of the families want to live in neighbouring plots, in the presented model there is an option for several families to submit a collective preference list. The model refers to this collection of families as one tenant. In the case where all the tenants are individuals, our assignment problem is the classic minimum cost perfect matching in bipartite graphs. There are several polynomial algorithms that are known for solving this problem [11,13]. In the general case, the assignment problem is a problem of finding a minimum perfect matching in a bipartite hypergarph, which is known to be NP-hard [3].

We carried out our procedure on a real community in Israel. The paper describes the setting of the problem, discusses the preparation procedures (establishing l), and provides the actual outcome. Finally, the paper shows the reactions of some members of the community after the allocation procedure.

The paper is built as follows: Section 2 introduces the model of the problem we are addressing. Section 3 describes the case study. Section 4, presents a discussion and provides our conclusions.

2 Problem Definition

The model consists of a set T of m tenants (i.e., clients or agents) and a set P of n plots (i.e., resources or objects). A tenant can represent a family or several families that want neighbouring plots. A tenant that is composed of a collection of k families is called a *multi-tenant* of size k or simply a k-tenant. A multi-tenant of size 1 is also called a *single tenant*.

Each k-tenant chooses l disjoint sets, each of size k. Each set is denoted $S_{i,q}$ where $i = 1, \ldots, m$ and $q = 1, 2, \ldots, l$. Namely, the set $S_{i,1}$ is chosen from P and is the set of k preferred plots for tenant i. The set $S_{i,2}$ is the set of preferred plots (for tenant i) taken from $P \setminus S_{i,1}$ and so on. In detail, the set $S_{i,q}$ is the set of preferred plots (for tenant i) taken from $P \setminus \bigcup_{t=1}^{q-1} S_{i,t}$. The list $(S_{i,1}, S_{i,2}, \ldots, S_{i,l})$ forms the *preference list* of tenant i.

The idea is to assign plots to the tenants such that each plot is assigned to at most one family. In detail, a k-tenant is assigned a set of k plots, and the assigned sets are disjoint.

A given assignment X sets for each tenant a rank r, which is defined as follows: If tenant i was assigned the set $S_{i,q}$, the rank will be q. If the assignment was of a set not in the preference list of the tenant, the rank will be $l + 1$. Explicitly, the rank of tenant i is

$$r_i(X) = \begin{cases} q & \text{if tenant } i \text{ was assigned with the set } S_{i,q} \\ l+1 & \text{otherwise} \end{cases}$$

A tenant desires to be assigned $S_{i,q}$ of the smallest possible q, i.e., to receive the lowest possible rank. To keep notation simple, when it is possible we refer to $r_i(X)$ simply as r_i.

The goal is for the assignment to maximise the satisfaction of the tenants. This is obtained by defining an objective function, $W(X)$, which we want to minimize. $W(X)$ is a sum of weights w_r^k, where each weight depends on the size k and the rank r of a tenant. Explicitly,

$$W(X) = \sum_{i=1}^{m} w_{r_i}^{k_i} \tag{1}$$

The weights w_r^k must obey the following conditions:

$$w_r^k < w_{r+1}^k \tag{2}$$

$$w_r^k + w_{r+1}^{k+1} < w_{r+1}^k + w_r^{k+1} \tag{3}$$

Condition 2 ensures that lower rank is preferable. Condition 3 gives an advantage to a k-tenant with fewer families. This is due to the fact that as the size of the tenant grows, the assigning process has more constraints. In addition, when some families choose to be a k-tenant, it is because they want to live next door to each other. Therefore, we assume that they will be willing to compromise on the rank, in order to be neighbours.

To simplify the weight function and still comply with conditions 2 and 3 we chose $w_r^k = \alpha_k w_r$. W.l.o.g one can choose $\alpha_1 = 1$, i.e., $w_r = w_r^1$. The new obtained conditions, corresponding to inequalities 2 and 3, respectively, are

$$w_r < w_{r+1} \tag{4}$$

$$\alpha_k > \alpha_{k+1} \tag{5}$$

There is a spectrum of ways to interpret the notion of maximising the satisfaction of the tenants. To further comprehend this, look at two assignments, X_1, X_2, which differ only in the plots assigned to two tenants of the same size. In X_1 both tenants have rank r, and in X_2 one has rank $r-1$ and the other $r+1$. The choice of which assignment is preferable is agenda-dependent. We describe two agendas for the problem. The first agenda is of a more socialistic nature, in which there is a preference for an assignment that gives tenants medium ranking plots over an assignment that gives some of the tenants high ranking plots and some of them low ranking plots. We refer to this as the *socialist agenda*.

The second agenda is of a more capitalistic nature, in which there is a preference for an assignment such that a group of tenants receives preferred plots even if it causes other tenants to be assigned less preferred plots. We refer to this as the *capitalist agenda*.

In both agendas, the optimal assignment should first minimize the number of tenants of rank $l+1$. Namely, an assignment that assigns to even one

tenant a plot not from its preference list, is less desired than any assignment that assigns only preferred plots. Formally,

$$w_{l+1} > m * w_l \tag{6}$$

An essential condition for the socialist agenda is:

$$2 * w_r < w_{r-1} + w_{r+1} \tag{7}$$

Similarly, an essential condition for the capitalist agenda is:

$$2 * w_r > w_{r-1} + w_{r+1} \tag{8}$$

Denote $\delta_r := w_{r+1} - w_r$. The socialist condition is

$$\delta_{r-1} < \delta_r \tag{9}$$

and the capitalist condition is

$$\delta_{r-1} > \delta_r \tag{10}$$

When $\delta_1 >> \delta_2 >> \ldots >> \delta_{l-1}$ the obtained objective function is one known in the literature as "rank maximal function" [9,10]. It maximizes the tenants receiving rank 1, then maximises the number of tenants receiving rank 2, and so on[1]. The other end of the spectrum is obtained by taking $\delta_1 << \delta_2 << \ldots << \delta_{l-1}$. This minimizes the number of tenants receiving rank l, then minimizes the number of tenants receiving rank $l-1$, and so on. This function is known as a "strong max-min function" [7].

We next show the relation between the proposed model and the problem of finding a minimal weighted perfect matching of a hypergraph. A hypergraph $H = (V, E)$ contains a set of vertices V and a collection E of subsets of V, called hyperedges. A hypergraph is ρ-uniform if all hyperedges contain exactly ρ vertices. A hypergraph can be weighted if each hyperedge is assigned a weight. A bipartite hypergraph is a hypergraph that partitions the vertices into two disjoint sets, L, R, such that each edge contains exactly one vertex from the set L and all other vertices from the set R. A perfect matching in a bipartite hypergraph is a collection of disjoint hyperedges such that each vertex in L belongs to exactly one hyperedge in the collection [2].

Our model can be viewed as a weighted bipartite hypergraph as follows: Consider the hypergraph $H = (V, E)$, where the set of vertices V is composed of two disjoint sets of vertices: The set L that contains a vertex for every tenant, and the set R that contains a vertex for each plot. Each weighted hyperedge, in H, represents a preference of a tenant. Specifically, for some k-tenant, and one of its preferences, the hyperedge contains the vertex that represents the specific tenant and the k vertices that represent the plots that assemble the preference. The weight of this hyperedge is w_q^k if it is the q-th preference ($1 \leq q \leq l$). If it is possible to assign to each tenant one of its preferences,

[1] In our work we first minimize the number of tenants of rank $l + 1$, as guaranteed by condition 6.

then the best assignment can be displayed by a collection of hyperedges of H. Because each tenant is assigned exactly one of its preferences, this collection is a minimal weight perfect matching.

The problem of finding a perfect matching in a ρ-uniform bipartite hypergraph is NP-complete, for $\rho \geq 3$ [3]. Hence, if there exist hyperedges that contain more than two vertices (i.e., the hypergraph is not a graph) the problem of finding a perfect matching in a nonuniform bipartite hypergraph is also NP-hard. It follows that finding the minimal perfect matching in a nonuniform bipartite hypergraph is NP-hard.

Notice, that in the case where there is no assignment that assigns to each tenant one of its preferences, there is a need to consider all relevant hyperedges with weights w^k_{l+1}. However, none of the fundamental concepts change. So, our problem is NP-hard.

3 Case Study

The work described in this article is based on the allocation of plots to families living in an Israeli community. In this chapter, we describe the characteristics of the specific problem and the corresponding results, including a satisfaction survey.

A new area was designated for house construction, divided into 44 plots that are partitioned into six areas. At the time of allocation, 34 families were registered. The neighbourhood committee conducted an initial survey in which each family was asked to choose three ranked options. A simple analysis of the results showed that a preference list of size three did not suffice; the number of plots obtained by unifying all the preferred plots was less than the number of families, i.e., for $l = 3$ there is no solution in which all families receive one of the three given preferences. In addition, the committee sought to provide each family with an option to choose together with another family. Such a choice guarantees that those two families will receive adjacent plots. The problem thus corresponds to a weighted matching problem in a bipartite-hypergraph with vertices representing the tenants on one side, vertices representing the plots on the other, and hyperedges of size two or three, corresponding to single-tenants and multi-tenants of size two, respectively.

The first issue that we dealt with was the determination of the appropriate value of l, i.e., the size of the preference list. From the initial three-option survey, we have learned that some areas are more desired then others. Based on this information, we have generated random problems, with a preference distribution similar to that of the pilot survey. The number of 2-tenants varied between 2 and 10. A simulated-annealing local search algorithm was used to solve the problem. In order to estimate l, we found the obtained ranks for each simulation. In all the generated instances, the worst rank was five. Out of an abundance of caution, we have set $l = 6$.

In Figure 1 we present the number of times each plot was selected, by all the tenants, in any of their six preferences. Five plots where not selected by

any tenants, so that the practical plot number is $n = 39$. The problem consists of thirty single-tenants and two 2-tenants, making a total number of $m = 32$ tenants. Each tenant request consists of a list of length $l = 6$. It is clear that some plots are more desirable than others, making the instance interesting.

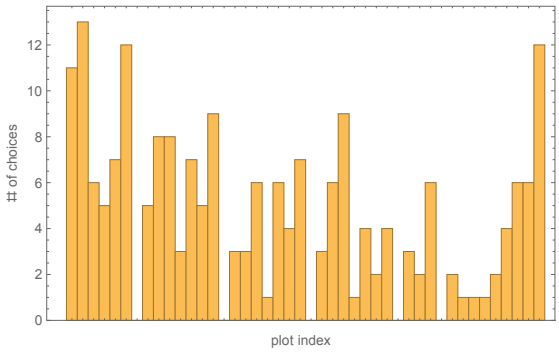


Fig. 1 The number of times each plot was selected, in any of the tenant's six preferences. Note that five plots were not selected.

The project committee chose the socialist agenda. There is an infinite number of functions that satisfy Equation 7 of the socialist agenda. We examined exponential functions of the form $w_r = a^r$ with parameter a. Choosing a higher value of a results in a more socialistic solution. To show that, consider X_1 and X_2, which are identical assignments, but for the following difference: In assignment X_1, $d+1$ tenants receive rank r. In assignment X_2, d tenants receive rank $r-1$, while one tenant receives rank $r+1$. Assignment X_2 is less socialistic in the sense that most tenants have improved their situation (compared to assignment X_1), but at the expense of one tenant. Which assignment has a lower objective function value? It is easy to see[2] that if $a > d$ the algorithm will prefer the more socialistic assignment, if $a < d$ the less socialistic assignment is preferred. When $a = d$, the two assignments have the same objective function value.

For our research study we also regarded the capitalist objective. It is easier to define weights w_r by setting the weight differences δ_r. We tried weight differences of the form $\delta_r = b^r$. As the value for b increases, the optimal assignment becomes more capitalistic.

The low number of 2-tenants make reaching an optimal solution an easy task. For each given preference that each of the 2-tenants gave in their preference list, the remaining problem is a weighted perfect matching problem in a graph, which can be solved to optimality in polynomial time. In accordance with the agenda of the project managers, we chose an exponential weight with a high value of a. Weights with $a \geq 3$ lead to the same solution, the "strong

[2] This follows from $W(X_1) - W(X_2) = (d + 1 - \frac{d}{a} - a)\ a^r$.

max-min" solution. Using this weight ensures that each tenant gets one of its first four given preferences.

Figure 2 is a histogram of ranks for several solutions: a socialistic objective with $a = 3$, which is a solution of the "strong max-min" objective and is the one that the real-life tenants got; a softer socialistic objective with $a = 2$; a soft capitalistic objective with $b = 8/7$ and a capitalistic objective with $b = 2$, which is the rank maximal solution. These four solutions are optimal for each given objective. The graph also presents the average number of tenants for each rank, when using a lottery procedure. In a lottery procedure, one generates a random order of the tenants. Then, each tenant, in its generated order, receives the best possible rank that is still available. In the lottery procedure the final result naturally depends on the generated order. In the figure, we show average results of 1000 different random ordered lists, generated with a uniform distribution. For tenants that could not accept one of the six options of their preference list, we chose random plots[3].

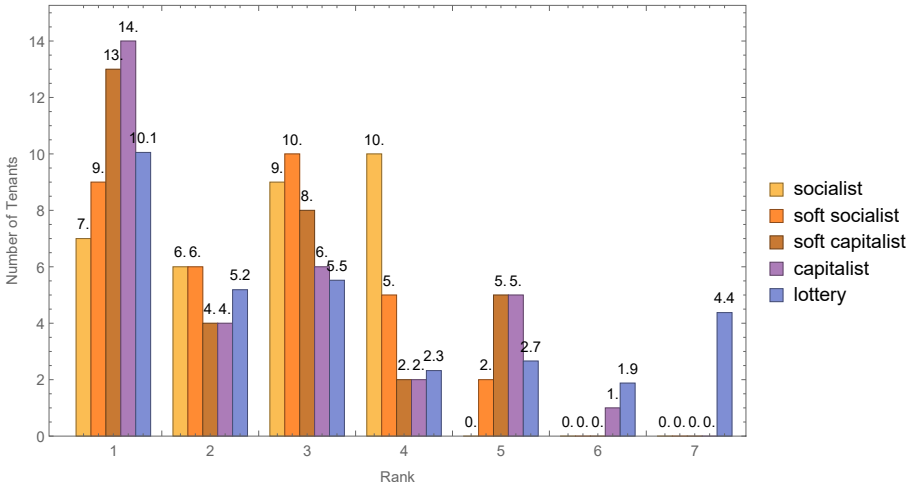


Fig. 2 A rank histogram for two socialist solutions, two capitalist solutions, and averaged results of a lottery procedure.

Figure 2 indicates that the lottery method is clearly inferior over the proposed method when comparing the received ranks. This statement can be quantified by calculating, for all the presented solutions, the average weighted rank $\frac{1}{m}\sum_{r=1}^{7} rN_r$, with N_r indicating the number of tenants receiving rank r, and the corresponding standard deviation. The numbers are presented in Table 1. Both the average rank and the standard deviation have the heighest value for the lottery procedure.

[3] In reality, the tenant chooses the most desired plot from what is left, but we had information only regarding the first six preferences

Solution	Socialist	Soft Socialist	Soft Capitalist	Capitalist	Lottery
Average Rank	2.69	2.53	2.44	2.47	3.17
Standard deviation	1.28	1.50	2.12	2.56	4.51

Table 1 The average rank and standard deviation for the solutions of Figure 2.

As part of our effort to understand satisfaction issues, we conducted a short survey among the tenants; 29 questionnaires were filled. Satisfaction was rated between 1 and 6, so that 6 expressed very high satisfaction and 1 expressed the lowest satisfaction. In Figure 3 we show the histogram of satisfaction levels. We divided the tenants into two groups; in one group, the tenants received ranks 1 or 2, and in the second group, they received ranks 3 or 4 (the size of the groups is 14 and 15, respectively). As expected, there is a correlation between the received rank and the level of satisfaction. Tenants that received ranks 3 or 4, were usually less satisfied. The correlation between the tenant rank and the corresponding satisfaction can be estimated according to Spearman's rank correlation coefficient, calculated taking a vector of obtained ranks and a vector of corresponding satisfaction levels. The calculated value [4] is (-0.5).

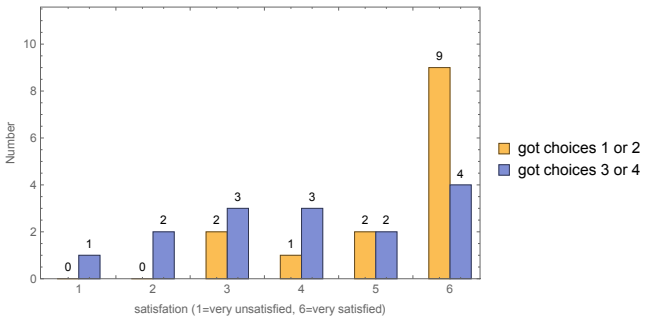


Fig. 3 A satisfaction level histogram. The tenants are separated to two groups; one receiving ranks 1 or 2, and the other receiving ranks 3 or 4.

As part of the survey, we asked the tenants to write personal comments. Some comments complimented the procedure, but we found the critical remarks more interesting. We end the case-study section by quoting several such remarks.

Three tenants wrote that $l = 6$ is too high. In their own words:

> "Six options is quite a lot. We are satisfied with what we have received but our 5-6 options was already a compromise."

[4] A value of -1 is obtained when there is a perfect anti-correlation between the vectors, and 0 value indicates no correlation.

> "We have not been able to find six locations that are equally good for us. The first three choices were good for us and although we had preferences, we lived with all three in peace. Priorities 4-6 were not very good for us."

> "The range of six options is problematic as options 1 and 2 are realistic, but the rest were given only because we had too."

These comments enhance the importance of setting properly the size of the preference list. Regarding the socialistic versus capitalistic agendas, we show a remark of one tenant that might prefer the capitalistic solution:

> "The distribution of the choices that you have presented to us[5] clearly shows that only few people received the first and second choices compared to many people who received the third and fourth. This form of ranking looked strange to us. We would expect the algorithm to give most of the first choices and a few last choices."

Mathematics, as apparent on Figure 2 and Table 1, shows the benefits of all the proposed solutions, both capitalistic and socialistic, over the lottery solution. Nevertheless, one tenant wrote:

> "As far as I was concerned, it was actually a lottery."

4 Discussion

In the article, we discussed the issue of matching plots to families, with a real-case reference. We have defined an objective function that allows finding Pareto solutions, laying on a spectrum that ranges between two ends: the maximal rank objective obtained by the strongest capitalist attitude, and a strong min-max objective of the strongest socialist attitude. Objective functions for each end of the spectrum were discussed individually [12, 7], but here we offer the possibility of obtaining the two spectrum ends, together with less extreme solutions. This might better suit the tenants and may increase their overall satisfaction.

From our experience with human nature, we expect that even the best solution will not lead to fully, and equal satisfaction. The capitalist solution increases the number of tenants who receive highly favored plots, but it also increases the standard deviation by letting some tenants receive less favored plots. What is better in terms of overall satisfaction remains an open question. The answer may depend on the specific case. It may be worthwhile to offer tenants a number of alternatives (anonymously) and let them, or a selected committee, make the decisions. The survey we conducted gives initial clues to the topic of satisfaction, but is very basic and does not purport to answer the question broadly. Wider surveys on similar projects, with a combination

[5] The tenants received a rank histogram of the received socialistic solution.

of researchers from softer research areas, can contribute to understanding this issue.

The paper deals with creating expectations among the families. It does so by establishing the proper length of the preference list, l. Specifically, if l is too small the expectation will probably not be met. However, setting l too large can give the families the feeling that their preferences are redundant. In the case study, l was established using simulations that were conducted before collecting the preferences. It will be interesting to explore mathematical methods that can help find the proper length of the preference list. Specifically, how likely is it that each tenant will get one of its l most preferred plots, which reduces to the question: how likely is it that a bipartite hypergraph with specific properties has a perfect matching? Such studies exist in cases where there are no multiple tenants, for example [6].

As far as the matching between the desired plots and those actually assigned is concerned, the lottery solution is unequivocally inferior to the proposed solutions. This diagnosis is reinforced by looking at the average rank and standard deviation shown in Table 1. Nevertheless, the lottery method has a certain advantage when acquaintance between the families exists, and especially for families choosing at mid-stages. Because the choice is public, considerations regarding potential neighbours are added to location preferences, making it possible to consider families with whom one does not want to be a neighbour with. The solution we proposed refers to the possibility of choosing neighbours that one wants to live next to, but not the negative possibility. In fact, in a meeting we held with the committee, the negative possibility did come up and eventually the committee rejected it. It is interesting to consider the algorithmic and computational implications of adding this possibility.

5 Acknowledgements

Our thanks to the community members of Kibbutz Hatzor for participating in this study; to “Korom” company for enabling the connection with the kibbutz; to Tal Grinshpoun and Yuval Ganot for their enlightening comments.

References

1. David J Abraham, Katarína Cechlárová, David F Manlove, and Kurt Mehlhorn. Pareto optimality in house allocation problems. In *International Symposium on Algorithms and Computation*, pages 3–15. Springer, 2004.
2. Ron Aharoni and Ofra Kessler. On a possible extension of hall’s theorem to bipartite hypergraphs. *Discrete mathematics*, 84(3):309–313, 1990.
3. Chidambaram Annamalai. Finding perfect matchings in bipartite hypergraphs. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1814–1823. SIAM, 2016.
4. Haris Aziz. Mechanisms for house allocation with existing tenants under dichotomous preferences. *arXiv preprint arXiv:1603.03073*, 2016.
5. Haris Aziz, Serge Gaspers, Simon Mackenzie, and Toby Walsh. Fair assignment of indivisible objects under ordinal preferences. *Artificial Intelligence*, 227:71–92, 2015.

6. Alan Frieze. Perfect matchings in random bipartite graphs with minimal degree at least 2. *Random Structures & Algorithms*, 26(3):319–358, 2005.
7. Daniel Golovin. *Max-min fair allocation of indivisible goods*. Citeseer, 2005.
8. Aanund Hylland and Richard Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political economy*, 87(2):293–314, 1979.
9. Robert W Irving. Greedy matchings. *Technical R eport Tr-2003-136, University of G lasgow*, 2003.
10. Robert W Irving, Telikepalli Kavitha, Kurt Mehlhorn, Dimitrios Michail, and Katarzyna Paluch. Rank-maximal matchings. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 68–75. Society for Industrial and Applied Mathematics, 2004.
11. László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.
12. Dimitrios Michail. Reducing rank-maximal to maximum weight matching. *Theoretical Computer Science*, 389(1-2):125–132, 2007.
13. Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

MISTA 2019

Aggregating partial orders - a case study in the internal logistics sector

Blandine Vacher, Antoine Jouglet, Dritan Nace, Stephane Pietrowicz and Marwane Bouznif

Abstract The focus of this work is the internal logistics sector of industry. The problem that we address involves aggregating a number of ordered flows of loads (totes, cartons, containers, etc.) into one single flow. It is an important problem that has applications in a variety of contexts. From the theoretical perspective it belongs to the category of partial order aggregation, known to be NP-Hard [1]. In this paper we recall the known problem and describe a use-case application in logistics. We then present two exact methods for aggregating loads, the first based on linear integer programming and the second using dynamic programming. Both methods perform very well for realistic industrial instances. Finally, we present a heuristic that we developed. This heuristic is able to give a solution very close to the optimal solution within one millisecond, and it has been adopted by the company behind this study. Numerical results will illustrate our findings.

Keywords Sorting, Partial orders, Dynamic programming, Optimization

Blandine Vacher
SAVOYE, Dijon 21000, France
E-mail: blandine.vacher@savoye.com

Antoine Jouglet
Sorbonne Universités, Université de Technologie de Compiègne, CNRS, Heudiasyc UMR 7253, CS 60319, Compiègne cedex 60203, France
E-mail: antoine.jouglet@hds.utc.fr

Dritan Nace
Sorbonne Universités, Université de Technologie de Compiègne, CNRS, Heudiasyc UMR 7253, CS 60319, Compiègne cedex 60203, France
E-mail: dritan.nace@hds.utc.fr

Stephane Pietrowicz
SAVOYE, Dijon 21000, France
E-mail: stephane.pietrowicz@savoye.com

Marwane Bouznif
A-SIS, St Etienne 42000, France
E-mail: marwane.bouznif@a-sis.com

1 Introduction

New technologies and an increasingly consumerist society are significantly impacting warehouse management, which includes supply, receipt, and flow and inventory management. This has forced the logistics sector to increase its reactivity and competitiveness. SAVOYE is a manufacturer of equipment for the preparation of orders, specialized in the automation and computerization of logistics warehouses. The company is concerned with evaluating and improving logistics solutions, which is our focus in this study.

1.1 Context

Improving delivery times means improving and optimizing both routing solutions and order picking, and to this end considerable resources are being invested in developing autonomous systems. *Automated Storage and Retrieval Systems* (ASRS) [2] are particularly worthy of attention. An ASRS consists of a storage area where responsibility for moving stock in and out is assigned to an automatic system (a crane or automated shuttle). ASRSs reduce human error and execution time, and positively impact the overall activity flow. They are often a component of *Goods-To-Person (GTP)* solutions. GTP solutions are global systems using an automated storage facility (such as an ASRS) to store items and a transport area for conveying loads (totes, cartons, containers, etc.) directly to a fixed workstation where the order can be fulfilled. SAVOYE uses its own GTP solution, *GTP Intelis PTS (Picking Tray System)*, that incorporates SAVOYE's *Intelis PTS* ASRS [3].

PTS ASRSs that correspond precisely to SAVOYE's design are not to be found in the literature. However, all ASRSs exhibit similar behavior and raise similar issues that need to be dealt with. Thus it is interesting to refer to the literature review on ASRSs prior to 2009 that is given in [2]. Roodbergen and Vis describe different types of ASRSs, including "unit-load", "mini-load", "man-on-board", "deep-lane", "flow-rack system" ASRSs [4]. New types of systems have continued to be created, such as "free-fall-flow-rack" ASRSs in 2008 described by [5], and SAVOYE's "Intelis PTS" ASRS that is our focus here. Intelis PTS belongs to the category of "Shuttle-based storage and retrieval systems", since it uses two different kinds of vehicles for horizontal and vertical movements respectively [6].

Figure 1 shows the complete GTP Intelis PTS system with Intelis PTS as the ASRS. It is important to maximize the throughput of loads within the global system so that orders can be completed quickly. It is not enough simply to maximize the exit flow, as the system needs to sort the loads to complete orders sequentially on GTP posts. The list of orders is fixed in advance, so the system knows in which order loads will arrive on each post. The company has decided not to sort loads inside the PTS aisles so as not to place constraints on throughput and storage capacity. Sorting is done between the storage exit and the GTP posts. Below we will present the pre-sorting that takes place between the conveyors awaiting the output loads from the PTS aisles and the collector.

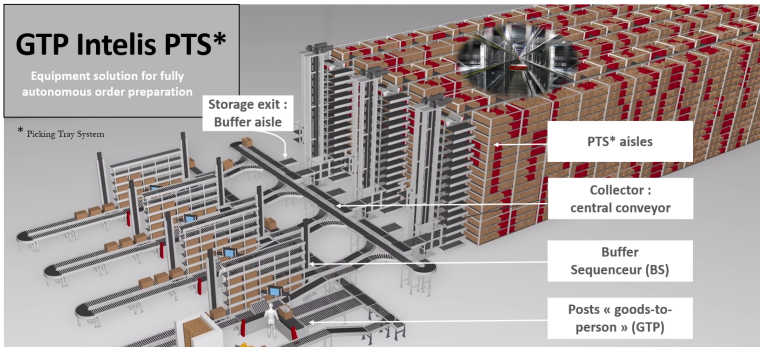


Fig. 1: A particular GTP ASRS : the GTP Intelis PTS

1.2 Calculation of an optimal aggregating sequence

The GTP solution presented in this paper comprises several elements as shown in Figure 1, namely PTS aisles, a collector, autonomous sorting systems known as "BSs", and GTP posts where orders are fulfilled. The PTS aisles forming the ASRS are storage aisles with elevators to deliver loads onto a collector. This collector links the storage part containing all loads with order boxes waiting on GTP posts. To complete an order, the required loads leave the storage facility with a preference number relating to an order of preference for processing (on GTP posts). This order of preference is not respected by the ASRS, i.e. loads exit the store as fast as possible, and not according to the order of preference.

Specifically, the PTS aisles are equally distributed along the collector. Each of them has its own exit, which is a buffer aisle materialized by a short conveyor leading to a point of access onto the collector. All (randomly) exiting loads will wait in line in their buffer aisle. The first load onto a buffer aisle can be injected onto the collector. The system has to know how to aggregate all these loads onto the same collector while respecting the order of preference as far as possible. This is in order to reduce reordering, which is done through an autonomous sorting system ("BS" in Figure 1) that receives this exit flow. The objective is to build a feasible aggregating sequence as near as possible to the order of preference. This aggregating sequence represents the collector exit flow, composed of all loads from buffer aisles once they are injected onto the collector. To the best of our knowledge, although the load sequencing problem [7,8] *inside the ASRS* has been studied in the literature, this is not the case for the load sequencing problem *at the ASRS exit*. Within the field of logistics it would appear to be a new problem, but it nevertheless corresponds to a known distance problem in the scientific literature, given that we are seeking to aggregate partial orders.

The paper is organized as follows. The mathematical distance problem corresponding to our problem is presented in Section 2 together with some known results from the state of the art. Section 3 presents the model for aggregating loads on the collector, which in logistics is an industrial use-case of the distance problem. Finally, methods are given as a mixed integer program, a dynamic program, and heuristics in Section 4, followed by a discussion and numerical results in Section 5.

2 Aggregating partial orders

The problem of aggregating partial orders from different sources appears in a number of fields, such as in voting theory [9]. We believe that internal logistics is a new application of this problem. In this section we present the mathematical model for the ranking problem and distance measures which can represent our logistics problem. Our inspiration for this was [10].

2.1 The distance problem

The problem is based on pairwise comparisons of elements to create a partial order. For two elements x and y, we have $x < y$ if x is preferred to y. Since this is a partial order, some elements x, z can be unrelated, and this is expressed as $x \sim z$.

The objective is to minimize the disorder between two orders. We use the Kendall tau distance K to compare them. For a given set L of candidates and two orders W and P, the rankings of candidate $x \in L$ is respectively given by $w(x)$ and $p(x)$. Then, the distance corresponds to the number of disagreements on the ranking of two candidates, $K(w,p) = |\{(x,y) : x,y \in L, w(x) < w(y)\ but\ p(x) > p(y)\}|$. Our problem is exactly equivalent to finding a linear extension σ of a given partial order W while minimizing the Kendall tau distance with a given order of preference P. We therefore use a version of the nearest neighbor Kendall tau distance K_N to compute the disagreements between all extensions of W respectively and the order P. One extension corresponds to a possible total order respecting all the partial order comparisons, and we denote the set of extensions of a partial order W as $EXT(W)$. Thus, we have $K_N(w,p) = min\{K(\sigma,p) : \sigma \in EXT(W)\}$.

Definition 1 The associated distance problem:
Instance: A set of candidates L, two (partial) orders W and P on L, and $k \in N$.
Question: Is $K_N(w,p) \leq k$, i.e. $\exists \sigma \in EXT(W)$ such that $K(\sigma,p) \leq k$?

From our logistics problem, P is a bucket order and W is a chain sum order. Elements in different buckets are totally ordered, whereas elements inside a bucket are neither related nor treated as equivalent. It will be remarked that we have instances where P becomes a total order. The chain sum order W consists of a set of k disjoint orders of length at most h. By definition, h is termed the height and k the width of W. Moreover, we suppose in our case that the $<$-relation is transitive.

2.2 Known complexity

We wish to give the total order extension of a given partial order W minimizing the differences with another given (partial) order P. This problem derives from the distance problem in [1, 10], and Brandenburg et al. give us complexity results.

When P is a total order and W a partial chain sum order of height $h > 11$, we know that our problem is $\mathcal{NP}$-*complete* and 2-*approximable*. It can be solved in polynomial time when $h < 3$. The nearest neighbor Kendall tau distance problem for two partial orders is known to be $\mathcal{NP}$-*complete* if at least one of them is a chain sum order of height at least 12. Finally, for two bucket orders the problem can be solved in $O(nlogn)$.

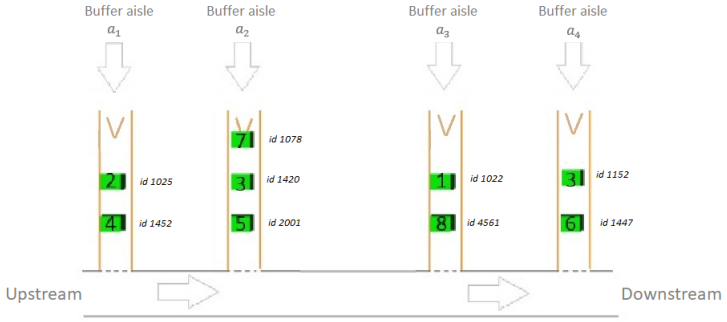


Fig. 2: Illustration of aggregating four ordered flows of loads onto a collector

3 Application of aggregating loads in logistics

Given the distance problem as defined above, this section gives the model used to address the load aggregation problem in logistics. Explanations and notations will be given to show how the two problems are linked.

We define a particular distance problem instance with a set of loads L. In this instance we are seeking to aggregate loads coming from several ordered lists (forming the partial order W) into a single total ordered list (best extension σ found for W) such that the order of preference (P) defined among loads is satisfied as fully as possible (measured in terms of K_N). The construction of the corresponding instance is presented below.

3.1 Notation

The set L contains all the n loads to be dispatched, each identified by a unique identifier. The system is assumed to have full knowledge of the loads present in each of the buffer aisles. Thus, the set $A = \{a_1, \ldots, a_k\}$ gives k buffer aisles numbered from the most upstream to the most downstream. We define $h_i = |a_i|$, the number of loads present in the buffer aisle a_i.

In each buffer aisle, loads are placed in a queue. For the purposes of notation, we define function $a_i : \mathbb{N} \to \mathbb{N}$ such that $a_i(j) = l \in L$ where l is the identifier of the j^{th} load waiting in aisle a_i. Loads in a same buffer aisle have to be injected according to where they are placed: that is to say, if load u is placed before load v in the same buffer aisle, then load u will necessarily be placed onto the collector before load v. These are called *precedence constraints* and represent a partial order W. In our case, these constraints correspond to k chains of parallel precedence constraints given by each aisle a_i and their transitive $<$ −relation $\{a_i(j) < a_i(j+1), \forall j \in [1, \ldots, h_i - 1]\}$. Thus W is specifically a chain sum order of length $h = max_{i \in [1,\ldots,k]} h_i$ and width k.

We denote as P the preferred order in L, with its associated ranking function $p : L \to \mathbb{N}$ such that, for any load $u \in L$, $p(u)$ is the preference exit order for p. That is, $\forall u \neq v \in L$, if $p(u) < p(v)$, it is preferable to place load u before load v. More than one load may have the same preference number, and thus p may express a bucket order. For two equal preferred numbers, it does not matter which load is placed first.

To illustrate this, Figure 2 shows an instance where $k = 4$, $n = 9$, $A = \{a_1, a_2, a_3, a_4\}$, $L = \{id1025, id1452, \ldots, id1447\}$. It can also be seen that $h_1 = 2$, $a_1(1) = id1452$, $a_2(1) = id2001$ and $p(id1452) = 4$. We have also the chain sum order according to the aisles $W = \{id1452 < id1025, id2001 < id1420 < id1078, id4561 < id1022, id1447 < id1152\}$ and, thanks to the ranking function p, the bucket order $P = \{id1022 < id1025 < id1420 \sim id1152 < id1452 < id2001 < id1447 < id1078 < id4561\}$.

3.2 Minimizing disorder

The aggregating sequence we are looking for is an extension of W, as precedence constraints have to be respected. Since our aim is to respect the order of preference as far as possible, we need to evaluate how far the aggregating sequence is from P, i.e. the disorder of the aggregating sequence. Thus, for any solution sequence $\sigma \in EXT(W)$, we measure the Kendall tau distance according to the ranking function p : $K(\sigma, p)$. We wish to find the best sequence σ, i.e. the one satisfying the smallest disorder $K_N(w, p) = min\{K(\sigma, p) : \sigma \in EXT(W)\}$. In Figure 2, one optimal solution, for instance, is given by $\sigma = \{id1452 < id1025 < id2001 < id1420 < id1417 < id1152 < id4561 < id1022 < id1078\}$ with a disorder (or distance) of 14.

We chose this definition of disorder because of the way installations are usually organized in practice, that is to say groups of buffer aisles that output loads into an autonomous sorting system [11]. It is important to reduce this disorder as much as possible, initially at the merging point, to increase the efficiency of the autonomous sorting system.

4 Solution methods

We solve the problem using both exact and heuristic methods. Below we detail a mixed-integer linear programming approach, followed by a dynamic programming approach, and finally two heuristic methods.

4.1 Integer Linear Programming

Our formulation of the problem as a mixed-integer linear programming (MILP) problem is as follows:

Parameters: Let L be a given set of n loads placed in k buffer aisles each of length h_i.

Variables: Let $x_{u,v}$ be binary variables such that $\forall (u, v) \in L^2, x_{u,v} = 1$ if u is placed before v in the aggregating sequence. The objective function minimizes the disorder defined in the previous section.

$$\min z = \sum_{(u,v)\in L^2 s.t. p(u)>p(v)} x_{u,v} \tag{1a}$$

$$x_{u,v} = 1, \forall u = a_i(j), v = a_i(l),$$
$$\forall i \in \{1,\ldots,k\} \text{ with } 1 \leq j < l \leq h_i \tag{1b}$$

$$x_{u,v} + x_{v,w} \geq x_{u,w}, \forall (u,v,w) \in L^3 \tag{1c}$$

$$x_{u,v} + x_{v,w} \leq 1 + x_{u,w}, \forall (u,v,w) \in L^3 \tag{1d}$$

$$x_{u,u} = 0 \tag{1e}$$

$$x_{u,v} + x_{v,u} = 1, \forall u \neq v, (u,v) \in L^2 \tag{1f}$$

$$x_{u,v} \in \{0,1\}, \forall (u,v) \in L^2 \tag{1g}$$

The objective function (1.a) represents the disorder defined in Section 3.1, i.e. the distance between the aggregating sequence and the ranking function p. The constraints (1.b) allow the precedence constraints W to be satisfied. Constraints (1.c) and (1.d) express transitivity inside order relations. Constraints (1.f) force some order between two different loads to give a total order.

4.2 Dynamic programming

The second approach is based on the dynamic programming paradigm. The idea is to separate the overall problem into sub-problems and then to recursively seek the optimal solutions to the sub-problems until all possibilities have been explored and a solution to the overall problem obtained. The main paradigm of dynamic programming is that each sub-problem has an optimal substructure in accordance with Bellman's "principle of optimality". We recall that our problem is to obtain an aggregating sequence that minimizes the disorder defined by the order of preference while respecting the precedence constraints established by loads in the buffer aisles. A sub-problem can be expressed as follows: what is the best aggregating sequence, with respect to the disorder defined above, for the set of loads composed of the first $(u_1, u_2, \ldots, u_k)$ loads in buffer aisles $(a_1, a_2, \ldots, a_k)$ respectively.

To devise a dynamic programming algorithm, we need two elements: the state of the system and a state function which gives the utility function corresponding to this state. In our problem we are given the configuration of buffer aisles and the loads present in each of them, which defines the initial state of the system.

Given an initial configuration defined by k buffer aisles and all loads of L placed in the buffer aisles, one state is represented by a k-uplet $u = (u_1, u_2, \ldots, u_k)$ with u_i indicating the number of loads being injected onto the collector from buffer aisle a_i. To each state we associate a function $f(u)$ that gives the smallest disorder attainable if loads are placed on the collector according to a particular aggregating sequence. The initial state u_0 is defined as k-uplet $(0, 0, \ldots, 0)$ and $f(u_0) = 0$. An instance of this is shown in Figure 3. It will be remarked that some general state u can be attained from k preceding states $(u_1 - 1, u_2, \ldots, u_k), (u_1, u_2 - 1, \ldots, u_k), \ldots, (u_1, u_2, \ldots, u_k - 1)$, and we have the following recursive formula:

$$f(u_1, u_2, \ldots, u_k) = \min\{f(u_1{-}1, u_2, \ldots, u_k), f(u_1, u_2{-}1, \ldots, u_k), \ldots, f(u_1, u_2, \ldots, u_k{-}1)\}.$$

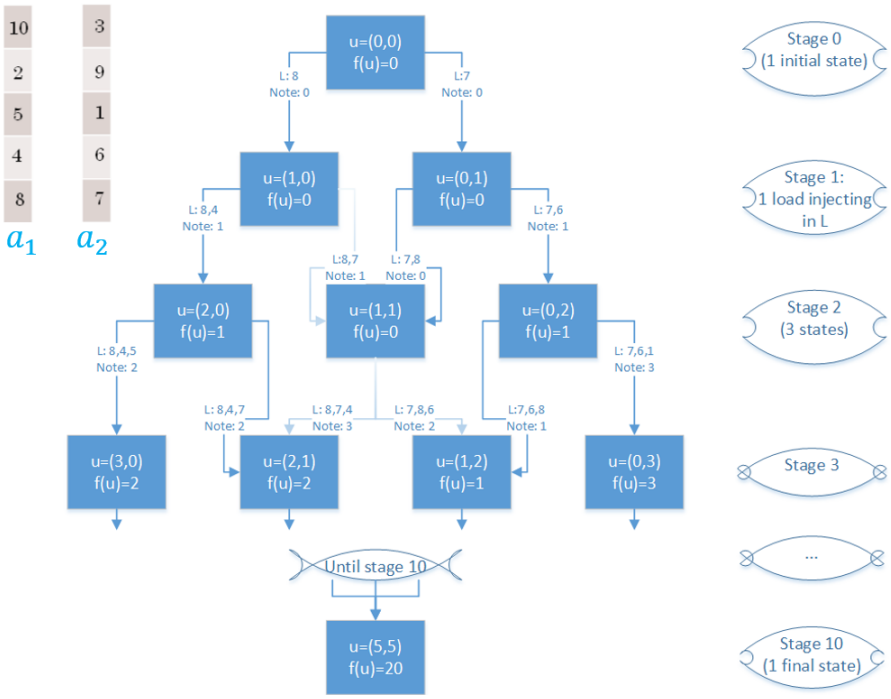


Fig. 3: Diagram of the dynamic programming through an example.

From the above it is straightforward to devise a dynamic programming algorithm exploring all states and achieving the optimal solution. The validity of the algorithm comes from the following obvious proposition.

Proposition 1 *Given an optimal aggregating sequence σ for state $u = (u_1, u_2, \ldots, u_k)$ where the last load in σ is from buffer aisle a_i, then the sub-sequence of σ that does not include this last load is optimal for state $u = (u_1, u_2, \ldots, u_i - 1, \ldots, u_k)$.*

The algorithm allows us to obtain the optimal sequence σ when reaching state $u = (h_1, h_2, \ldots, h_k)$. This may be done via a backtrack to preceding states, which can provide $f(u)$ from the final state through to the initial state. Alternatively, it may be done by keeping track, for each state u, of the preceding state (or the corresponding buffer aisle number) used in minimizing function $f(u)$.

To summarize, the idea of the algorithm is to retain all the best states at each step until a single final state is reached at the final stage. This algorithm has a complexity in $O(n^k)$. The problem is thus polynomially solved for a fixed number k of buffer aisles.

4.3 Approximated methods

We also developed two heuristic approaches to solve our problem. The first is a greedy-like heuristic that creates an ordered list of loads σ by adding loads one by one, without coming back on a decision already made. Thus, loads are added consecutively to the final list according to a criterion minimizing the disorder locally.

For our requirements, we tested different options:

- Option H_1: choose the load with the smallest preference number from among the first loads in the different buffer aisles;
- Option H_2: choose the first load in the buffer aisle that has the load with the smallest preference number among the first and second loads in the different buffer aisles;
- Option H_{all}: choose the first load in the buffer aisle that has the load with the smallest preference number among all the loads in the different buffer aisles;

Using one of these options we can create a sequence σ of loads. Calculating the disorder of this sequence gives a solution to our problem. We can use it to obtain an upper bound for our instance.

4.4 A dynamic programming based heuristic

Using the dynamic programming approach described above we developed an efficient heuristic.

At each iteration of the algorithm, two buffer aisles are chosen to be "merged". They constitute a sub-problem to be solved by the dynamic programming algorithm. The obtained exit sequence constitutes a new buffer aisle that replaces the two previous buffer aisles in a new problem. We continue to do this until only one buffer aisle remains. The sequence order for this final buffer aisle gives the aggregating sequence that we sought.

In this way, computation time is very short and our experiments showed that the solutions obtained are close to the optimal solution. This is the method that the company currently implements in its warehouses.

5 Numerical results

We tested all methods on the same randomly generated real-size instances ranging from five to seven buffer aisles with different number of loads on them. We tested ten random instances for each size. The obtained results are reported in the following tables.

Table 1 gives the average execution time of the exact methods: the MILP and the "ProgDyn" dynamic programming approach. The MIP has been solved with CPLEX version 12.8.0 and ProgDyn has been developed using C++.

Table 2 gives the average percentage gap with respect to the optimal solutions given by the exact methods ($Exact$). All results were obtained in less than 2 milliseconds. H_1, H_2 and H_{all} correspond to the basic heuristic with respectively one, two, or all levels, and "P.D.F." (for "ProgDynFusion") corresponds to the best heuristic based on the dynamic programming.

The results show that heuristics are instantaneous and "ProgDynFusion" finds solutions whose order differs by less than 0.2% from the optimal solutions.

In real-world situations there are between two and ten parallel buffer aisles with a capacity of no more than ten loads. Thus, industrial instances encountered in practice will not exceed 100 loads. Since it is a dynamic problem, the algorithms need to be run several times. For online use, around thirty loads at least need to be merged every minute. It follows from this that the performance of the dynamic programming

Table 1: Execution time of different exacts methods, under the same environment.

	Instances with buckets	**k=5 n=60 h=12**	**k=6 n=60 h=10**	**k=6 n=90 h=15**	**k=7 n=90 h=13**
	Methods	time (s)	time (s)	time (s)	time (s)
1	*MIP*	0.4862	0.4932	2.0822	1.9783
	ProgDyn	0.2845	1.7144	25.9369	206.346
2	*MIP*	0.5018	0.5179	2.032	1.9883
	ProgDyn	0.305	1.8979	24.2399	227.4337
3	*MIP*	0.5413	0.547	2.0711	2.0921
	ProgDyn	0.2865	1.4303	28.1447	208.0353
4	*MIP*	0.5576	0.5819	2.1216	2.1012
	ProgDyn	0.2919	1.6007	26.2396	235.1652

Table 2: Best disorders (D) obtained using different heuristics under the same environment.

	Instances with buckets	**k=5 n=60 h=12**	**k=6 n=60 h=10**	**k=6 n=90 h=15**	**k=7 n=90 h=13**
	Methods	D	D	D	D
1	Exact	608.8	593	1454.1	1405.3
	H_1	0.245	0.166	0.198	0.199
	H_2	0.100	0.103	0.118	0.116
	H_{all}	0.152	0.144	0.175	0.159
	P.D.F.	0.002	0.001	0.001	0.001
2	Exact	618.4	611.8	1412.9	1406.2
	H_1	0.181	0.219	0.208	0.164
	H_2	0.073	0.119	0.098	0.067
	H_{all}	0.168	0.181	0.166	0.186
	P.D.F.	0.002	0.004	0.001	0.001
3	Exact	603.3	566	1462.5	1423.1
	H_1	0.199	0.243	0.1804	0.217
	H_2	0.097	0.096	0.1076	0.088
	H_{all}	0.204	0.271	0.1905	0.185
	P.D.F.	0.001	0.002	0.0008	0.002
4	Exact	606.6	545.2	1408.1	1421.4
	H_1	0.190	0.231	0.191	0.156
	H_2	0.107	0.106	0.108	0.099
	H_{all}	0.190	0.159	0.168	0.186
	P.D.F.	0.000	0.001	0.002	0.001

approach described makes it suitable for no more than seven buffer aisles, while the heuristic based on the dynamic programming is suitable for all the company's required size instances.

The "ProgDynFusion" algorithm is therefore the best method for the company, since reducing the execution time is more important than the accuracy of the minimum disorder. It is also a method that is able to solve large-scale partial order aggregation problem really efficiently.

6 Conclusion

This paper presents an efficient method for aggregating loads in a complex logistics system (an archetypal *system of systems*) used by the logistics company SAVOYE. The problem of sequencing loads is solved using an integer linear programming approach and a dynamic programming approach. A heuristic based on the dynamic programming provides the company with its own solution without using a solver, where loads are pre-sorted in less than a millisecond and the resulting order remains very close to the optimum. We performed simulations to demonstrate significant improvements in the performance of the logistics system. This work has led to a patent [12].

References

1. Franz J. Brandenburg, Andreas Gleissner, *Ranking chain sum orders* (2016), Theoretical Computer Science, University of Passau, 94032 Passau, Germany.
2. Roodbergen, K.J. and Vis, I.F.A. (2009), *A survey of literature on automated storage and retrieval systems*. European Journal of Operational Research 194(2), 343-362.
3. Savoye, 2018. Goods-to-Person (Goods-To-Man) picking: Intelis PTS [www Document]. Intelis. URL http://www.intelis-equipment.com/en/our-solutions/goods-person-goods-man-picking-intelis-pts (accessed 6.26.18).
4. Amine Hakim Guezzen, Zaki Sari, Pierre Castagna, and Olivier Cardin *Travel Time modeling and simulation of a mobile racks automated storage/Retrieval System*
5. Dhiyaeddine Metahri, Khalid Hachemi, *Retrieval-travel-time model for free-fall-flow-rack automated storage and retrieval system*, Journal of Industrial Engineering International, 2018, Volume 14, Issue 4, pp 807-820.
6. Kalyanaraman, P., Keerthika, C., *A Review on Automated Storage/ Retrieval Systems and Shuttle Based Storage/Retrieval Systems*, International Journal on Recent and Innovation Trends in Computing and Communication, ISSN: 2321-8169, Volume: 4, Issue: 11, 167 171, 2016
7. Yugang Yu and Rene B.M. De Koster, *Sequencing heuristics for storing and retrieving unit loads in 3D compact automated warehousing systems*, IIE Transactions, 2011.
8. Min-Hong Han, Leon F. McGinnis, Jin Shen Shieh, John A. White, *On Sequencing Retrievals in an Automated Storage/Retrieval System*, School of Industrial and Systems Engineering, Atlanta, Georgia.
9. Conitzer V., Davenport A. and Kalagnanam H., *Improved bounds for computing Kemeny rankings* (2006), In Proc. of 21stAAAI, 620-626 21st AAAI, 620626.
10. Franz J. Brandenburg, Andreas Gleissner, *Comparing and Aggregating Partial Orders with Kendall Tau Distances* (2011), Technical Report, Number MIP-1102, University of Passau, 94032 Passau, Germany.
11. S. Pietrowicz, J-M. Collin, B. Vacher, *Procédé de pilotage en mode tri d'un système de stockage tampon et de séquencement de charges, et unité de pilotage correspondante* Référence brevet R34407FR (2018).
12. J-M. Collin, B. Vacher, A. Jouglet, D. Nace, *Procédé de séquencement de charges dans un système de distribution automatisé, avec réduction d'un désordre lors d'une collecte de charges sur un collecteur* Référence brevet R34848FR (2019).

MISTA 2019

Radiotherapy scheduling problem

Tu-San Pham · Louis-Martin Rousseau · Patrick De Causmaecker

Abstract Radiotherapy treatment (RT) is a popular form of cancer treatment where a patient receives a large amount of radiation over a course of several consecutive days. In the most common form of radiotherapy, the radiation is delivered by a linear accelerator (linac). Planning the usage of linacs is essential in reducing the waiting time of patients, which directly affects the treatment outcomes. We study the radiotherapy scheduling problem (RTSP), which consists of finding the treatment schedule for a set of patients given a set of linacs respecting many side constraints and preferences. We propose a two-phase approach for the problem. In the first phase, patient treatments are assigned to specific linacs and days. The second phase then decides the sequence of patients on each day/linac and the specific appointment times. For the first phase, a Mixed Integer Linear Programming (MILP) model is proposed and solved using CPLEX. For the second phase, a MILP and a Constraint Programming (CP) model are proposed. The test data is generated based on real data from the cancer center of CHUM - a large cancer center in Montréal, Canada with about 3500 new patients and 40.000 treatments a year. The results show that the CP approach outperforms MILP for this problem and is able to solve real-sized instances.

Tu-San Pham
KU Leuven
E-mail: san.pham@kuleuven.be

Louis-Martin Rousseau
Polytechnique Montréal

Patrick De Causmaecker
KU Leuven

1 Introduction

The number of cancer cases has been increasing sharply in the last few decades. According to the World Health Organization (WHO) [1], cancer is the second leading cause of death globally in 2018. It accounts for about 1 in 6 deaths. In particular, in Canada nearly 1 in 2 Canadians is expected to be diagnosed with cancer in their lifetime and about 1 in 4 Canadians to die from cancer [1]. The treatment facilities and personnel, however, do not grow proportionally, which not only causes lots of distress to the treatment facilities and staff but also results in long waiting for patients. The long waiting time (also referred as "killing time") of treatments having negative impact on the clinical outcomes has been proven through many articles [3, 8, 2]. Therefore, better planning to reduce waiting times is crucial in improving cancer treatment results.

Before deciding a treatment plan, a set of tests and imaging need to be done on the patient, which might include x-rays, CT scans, MRI scans, PET scans, etc. The treatment team then carefully makes a treatment plan for each patient based on his/her diagnosis. There are many options for treatments, e.g. surgery, chemotherapy, radiotherapy. Sometimes different kinds of treatment are combined together on one patient and can be repeated as needed. This work focuses on the radiotherapy treatment (RT) where a patient receives a high dose of radiation to kill cancer cells. The radiation is usually divided into a number of smaller doses called **fractions** to allow healthy cells to recover between treatments. A series of fractions is delivered during a course of several consecutive days with breaks in the weekends. In the most common form of radiotherapy, the radiation is delivered by a *linear accelerator* (linac).

The waiting time of RT treatment depends highly on the planning of linac usage. According to Canadian Institute for Healthy Information, in 2018 50% of Canadian patients started treatments within 8 days, 90% started treatments within 20 days and 97% started treatments within the recommended maximum 28 days (see figure 1). Those numbers are getting worse in the recent years (see figure 2) due to the increasing number of patients. Better planning in treatments is hence vital to reduce the "killing time"

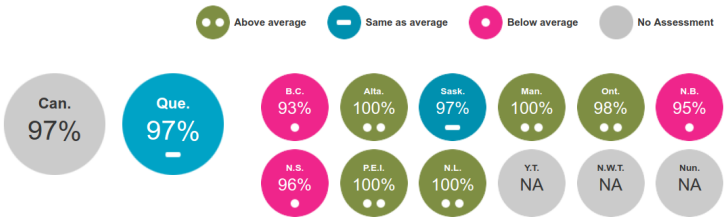


Fig. 1: Percentages of patients receiving treatments within the recommended deadline by states in Canada (2018)[2]

In this paper, we investigate the Radiotherapy Scheduling Problem (RTSP), which consists of deciding the radiotherapy treatment schedule for a set of patients in a horizon given a

[1] https://www.who.int/news-room/fact-sheets/detail/cancer
[2] https://yourhealthsystem.cihi.ca
[3] Canadian Institute for Healthy Information

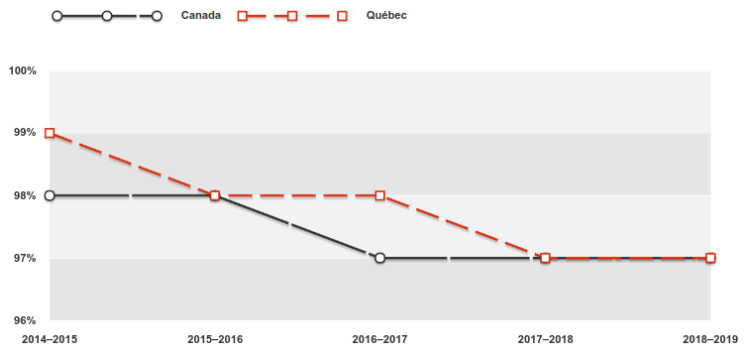


Fig. 2: Percentages of patients receiving treatments before the recommended deadline in Quebec [3]

set of linacs. Many constraints have to be considered, e.g. patients' due days, linacs' capacity, patients' preferences on appointment times, consistency of treatment in term of patient-linac assignments (i.e. a patient should not switch between linacs during his/her treatment period) and appointment times (i.e. a patient's appointment times should be consistent during his/her treatment period). Currently, in practice, most centers still create their appointment schedules by hand, which leaves lots of room for improvement. The problem, however, is complicated due to several strategies and constraints varying from center to center, which results in a large number of variants of the RTSP. In this paper, we consider a real-world variant of the RSTP present in the cancer center of CHUM (Centre hospitalier de l'Université de Montréal), a large cancer center in Montréal, Canada. Nevertheless, we aim to make the model as general and extensible as possible by taking into account many constraints and considerations.

We propose a two-phase approach for the problem. In the first phase, patient treatments are assigned to specific days and linacs. The second phase then decides the sequence of patients on each day/linac and the specific appointment times. For the first phase, a Mixed Integer Linear Programming model is proposed and solved using CPLEX. For the second phase, a Constraint Programming model is proposed and solved using CPOptimizer. We generate realistic instances based on real data from CHUM.

Radiotherapy scheduling has been studying for more than 20 years, started with the first two mathematical models of Conforti [4]. The objective of the models is to maximize the number of treated patients in a given horizon with the assumption that all treatments have a fixed duration of 15 minutes. The models are then extended in [5] to tackle more realistic instances where the treatment times of patients vary. In [15], a mathematical programming model is proposed to tackle the time preferences of patients. In [14], the radiotherapy scheduling problem is modelled as a discounted Markov decision process to provide a dynamic policy to take into account future events. In [13], the same authors extend their previous work to incorporate stochastic service times and perform a numerical analysis on a number of scenarios. Heuristics and metaheuristics are also applied widely to tackle large size instances. In [12,11], some constructive approaches and a GRASP-based algorithm are proposed. In [6], the authors propose four heuristic methods to solve a problem which consists of both pre-treatment and treatment stages. Similarly, heuristic approaches are also used in [10,9,16]. Each of the above-mentioned work, however, works with a different set of constraints and assumptions.

The contribution of this paper are three-fold. First, we introduce a general model, which is based on the model in [15] but is extended with a wide range of constraints and objectives. Our model tackles many constraints that are not considered together in many previous works, for example, the consistency in patient-linac assignments, or the changes in appointment times of fixed patients from the previous planning periods. Second, to the best of our knowledge, this is the first time constraint programming is successfully applied to the RTSP. Third, we describe in this paper how realistic instances are generated based on real data. In real life, the planning is usually made on a partly-full schedule which is filled with appointments from the previous planning periods. The schedule's layout and characteristics affect deeply the algorithm's performance. Hence it is important to test the algorithms on close-to realistic instances. Most works in literature either test their algorithms on empty instances with no fixed appointments or do not mention how their test instances are generated.

The structure of the paper is as follows. The problem description along with the mathematical model are given in section 2. Section 3 presents a two-phase model for the problem while data descriptions and numerical results are presented in section 4. Section 5 closes the paper with conclusions and future work.

2 Problem description

The radiotherapy scheduling problem (RTSP) consists of finding the schedule of treatments for a set of patients $\mathcal{P}$ over a planning horizon $\mathcal{T}$ given a set of linacs $\mathcal{K}$. Each patient $i \in \mathcal{P}$ has a release day r_i, which is the day that the patient is ready for treatment; a due day d_i, before which the treatment must start. A patient's due day is set based on his/her priority. Patients are categorized into palliative and curative. Palliative patients usually need urgent treatments within 1 to 3 days to relieve intense pain while curative patients can wait for longer to receive their treatments. Each patient i is associated with a treatment plan, which specifies the number of fractions I_i that the patient needs to receive, and the duration of each fraction (p_i) and the eligible linacs. All fractions of a patient have identical length, which is a multiple of 5, 10 or 15 minutes. In this paper, we assume the fraction duration is a multiple of 5 minutes and hence is measured in blocks of 5 minutes. A patient receives one fraction per day and the treatment must be carried out daily. Each treatment plan is also associated with one or several eligible groups of linac. Patients can switch between linacs of the same subgroup during the course of treatment. The total number of linacs assigned to a patient should be minimized, due to several reasons: (1) each linac is usually associated with a set of technicians which will get familiar with patients' condition before their treatment; (2) some patient-specified set up might have to be performed before each treatment, which leads to more time wasted if the treatment is carried out in many different rooms. Each linac has a daily capacity of C time slots. Some patients have preferences for their appointment times, usually in the early morning or late in the afternoon. The preference of patient i is indicated by a time window $[t_i^{min}, t_i^{max}]$, within which the appointment should take place. For a patient's convenience, his/her appointment times should be consistent during the course of treatment. The time preference and appointment time consistency, however, only apply for curative patients. Finally, in practice, the fixed appointments of patients from the previous periods sometimes can be moved to make space for new patients, in case there are several small intermittent available slots on the linac. However, the changes in appointment should be minimized.

The problem has many objectives, among which the most important one is reducing the waiting time. As analyzed before, the earlier the treatment starts, the better it is for their condition. Therefore, the treatment should be scheduled as soon as possible once the patient is ready. Other objectives includes: patient-linac consistency; appointment time consistency; preference time window for curative patients; and finally changes in appointment times of fixed patients.

We hereby present a Mixed Integer Linear Programming (MILP) model for the RTSP.

Parameters:

- $\mathcal{P}$: the set of patients, which consists of two subsets of palliative ($\mathcal{P}^P$) and non-palliative patients ($\mathcal{P}^N$). $\mathcal{P}$ is also divided into two subsets of new patients ($\hat{\mathcal{P}}$) and fixed patients ($\bar{\mathcal{P}}$) from the previous planning period.
- $\mathcal{T}$: the set of days in the horizon.
- $\mathcal{S}$: the set of time slots of each day.
- $\mathcal{K}$: the set of linacs.
- $\mathcal{K}^i$: the set of eligible linacs of patient i.
- C_k^t: the capacity of linac k in day t.
- r_i: the release day of patient i.
- d_i: the due day of patient i.
- p_i: the fraction duration for patient i.
- I_i: the number of fractions of patient i.
- $[t_i^{min}, t_i^{max}]$: the preference time window of patient i.
- $\bar{s}_i^t$: the appointment time slot of the fixed patient i on day t from the previous period.

Variables:

- x_{is}^{tk}: binary variables, is equal to 1 if patient i is assigned to linac k at time slot s of day t.
- z_i^t: binary variables, is equal to 1 if patient i receives his/her first treatment on day t.
- w_{ik}: binary variables, is equal to 1 if patient i is assigned to linac k during his/her treatment.
- $\bar{\Delta}_i^t$ and $\underline{\Delta}_i^t$: continuous variables, represent deviation from the preference time window of the new, non-palliative patient $i \in \mathcal{P}^N$ on day t (sooner or later).
- $\bar{\theta}_i$ and $\underline{\theta}_i$: continuous variables, represent the earliest and latest appointment time slots of the new, non-palliative patients $i \in \mathcal{P}^N$.
- $\bar{\phi}_i^t$ and $\underline{\phi}_i^t$: continuous variables, represent changes in appointment times (to sooner and later) of the fixed patient $i \in \bar{\mathcal{P}}$ on day t.

Model:

$$\text{minimize } \omega_1 \sum_{i\in P}\sum_{t\in T} t z_i^t + \omega_2 \sum_{i\in P}\sum_{k\in K} w_{ik} + \omega_3 \sum_{i\in\mathcal{P}^N, t\in T} (\bar{\Delta}_{it} + \underline{\Delta}_{it}) + \omega_4 \sum_{i\in\mathcal{P}^N} (\bar{\theta}_i - \underline{\theta}_i) + \omega_5 \sum_{i\in\mathcal{P}, t\in T} (\bar{\phi}_i^t + \underline{\phi}_i^t) \quad (1)$$

subject to

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}} x_{is}^{tk} \le 1, \qquad \forall i \in \mathcal{P}, t \in \mathcal{T} \quad (2)$$

$$\sum_{i\in\mathcal{P}} x_{is}^{tk} \le 1, \qquad \forall k \in \mathcal{K}, s \in \mathcal{S}, t \in \mathcal{T} \quad (3)$$

$$x_{is}^{tk} = 0, \qquad \forall i \in \mathcal{P}, t \in \mathcal{T}, k \notin \mathcal{K}^i, s \in \mathcal{S} \quad (4)$$

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}}\sum_{t\in\mathcal{T}} x_{is}^{tk} \le I_i, \qquad \forall i \in \mathcal{P} \quad (5)$$

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}}(x_{is}^{tk} - x_{is}^{t-1,k}) \le \sum_{k\in\mathcal{K}, s\in\mathcal{S}} x_{is}^{nk}, \qquad \forall i \in \mathcal{P}, t \in \mathcal{T}, \; n = t+1, \ldots, \min\{|\mathcal{T}|-1, t+\mathcal{I}_i - 1\}, \; \{x_{is}^{-1,k} = 0, \forall i, k, s\} \quad (6)$$

$$\sum_{k\in\mathcal{K}}\sum_{s\in\mathcal{S}}\sum_{t=0}^{d_i} x_{is}^{tk} \ge 1, \qquad \forall i \in \mathcal{P} \quad (7)$$

$$x_{is}^{tk} = 0 \qquad \forall i \in P, k \in K, s \in S, t \in \{0, \ldots, r_i\} \quad (8)$$

$$x_{is}^{tk} \le 1 - x_{i's'}^{tk}, \qquad \forall i, i' \in \mathcal{P}, k \in \mathcal{K}, t \in \mathcal{T}, \; s = 0, \ldots, |\mathcal{S}| - p_i + 1, \; s' = s+1, \ldots, s+p_i-1 \quad (9)$$

$$x_{is}^{tk} = 0, \qquad \forall t \in \mathcal{T}, i \in \mathcal{P}, k \in \mathcal{K}, \; s = |\mathcal{S}| - p_i + 1, ...|\mathcal{S}-1| \quad (10)$$

$$z_i^t \ge \sum_{k\in K}\sum_{s\in S} x_{is}^{tk} - \sum_{k\in K}\sum_{s\in S} x_{is}^{t-1,k}, \qquad \forall i \in P, t \in T, x_{is}^{-1,k} = 0 \quad (11)$$

$$w_{ik} \ge x_{is}^{tk} \qquad \forall i \in P, t \in T, k \in K, s \in S \quad (12)$$

$$\bar{\Delta}_i^t \ge x_{is}^{tk}(t_i^{min} - s), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}^N, s \in \mathcal{S} \quad (13)$$

$$\underline{\Delta}_i^t \ge x_{is}^{tk}(s - t_i^{max}), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}^N, s \in \mathcal{S} \quad (14)$$

$$\bar{\theta}_i \ge \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^N, k \in \mathcal{K} \quad (15)$$

$$\underline{\theta}_i \le \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^N, k \in \mathcal{K} \quad (16)$$

$$\bar{\phi}_i^t \ge \bar{s}_i^t - \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}} \quad (17)$$

$$\underline{\phi}_i^t \ge \sum_{s\in\mathcal{S}} s x_{is}^{tk} - \bar{s}_i^t, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}} \quad (18)$$

Objective function (1) minimizes the following objectives: (i) the waiting days of patients; (ii) the number of linacs assigned for each patient; (iii) the deviations from the preference time windows; (iv) the inconsistency in appointment times; and (v) the changes in appointment times of fixed patients. ω_1 to ω_5 are the weights of those objectives respectively. Constraints (2) ensure that each patient is scheduled at most one session per day. Constraints (3) ensure that at most one patient is scheduled for each slot on each linac per day. Constraints (4) prevent assigning patients to non-eligible linacs. Constraints (5) make sure each patient

is scheduled less than the required number of sessions. Constraints (6) take care of the continuity of treatment (daily treatments) until the end of the horizon or until the required number of sessions met. Constraints (7) and (8) make sure all patients are scheduled before their due day but after their release day. Constraints (9) ensure the number of required time slots for each patient by blocking the next slots on the linacs. Constraints (10) ensure no patients are scheduled on the last slot(s) of the days which are not sufficient for their fraction. Constraints (11) define the first days of treatment of patients. Constraints (12) are linking constraints between w and x. Constraints (13) and (14) define the deviation from the preference time window. Constraints (15) and (16) define the earliest and latest appointment time of a patient during his/her treatment course. Constraints (17) and (18) define the changes in appointment times of fixed patients

3 A two-phase approach

In practice, minimizing the waiting time of patients is the most important objective and is considered foremost before any other preferences in many cancer centers. This can be simulated in the above model by setting the weight of the first term of the objective function (ω_1) much larger than the weights of the remaining terms. When ω_1 is large enough, solving the problem is equivalent to solving two sub-problems sequentially: (1) determining the assignments of fractions to days and linacs; (2) given the assignments patient-day-linac, deciding the sequence of patients and the exact appointment times. We hereby propose separate models to solve these two phases sequentially.

3.1 Phase 1: assigning patients to days and linacs

Phase 1 decides the starting days of patients and also assigns the treatments to specific linacs. We propose a MILP model for phase 1. This model only works on the set of new patients $\hat{\mathcal{P}}$. The following (binary) variables are introduced.

- y_{ik}^t = 1 if the new patient i is assigned to day t on linac k, 0 otherwise.
- z_i^t = 1 if the new patient i receives his/her first treatment on day t, 0 otherwise.
- w_{ik} = 1 if the new patient i is assigned to linac k during his/her treatment.

Model:

$$\text{minimize } \omega_1 \sum_{i\in\hat{\mathcal{P}}}\sum_{t\in T} t z_i^t + \omega_2 \sum_{i\in\hat{\mathcal{P}}}\sum_{k\in K} w_{ik} \tag{19}$$

subject to

$$\sum_{k\in K} y_{ik}^t \le 1 \qquad \forall i\in\hat{\mathcal{P}}, t\in T \tag{20}$$

$$y_{ik}^t = 0, \qquad \forall i\in\hat{\mathcal{P}}, t\in\mathcal{T}, k\notin\mathcal{K}^i \tag{21}$$

$$\sum_{k\in K}\sum_{t\in T} y_{ik}^t \le I_i \qquad \forall i\in\hat{\mathcal{P}} \tag{22}$$

$$\sum_{k\in K}(y_{ik}^t - y_{ik}^{t-1}) \le \sum_{k\in K} y_{ik}^n \qquad \forall i\in\hat{\mathcal{P}}, t\in T,$$
$$n = t+1,\dots,min\{|T|-1, t+I_i-1\}, y_{ik}^{-1}=0 \tag{23}$$

$$\sum_{k\in K}\sum_{t=0}^{d_i} y_{ik}^t \ge 1 \qquad \forall i\in\hat{\mathcal{P}} \tag{24}$$

$$y_{ik}^t = 0 \qquad \forall i\in\hat{\mathcal{P}}, k\in K, t\in\{0,\dots,r_i\} \tag{25}$$

$$\sum_{i\in\hat{\mathcal{P}}} p_i \times y_{ik}^t \le \hat{C}_k^t \qquad \forall t\in T, k\in K \tag{26}$$

$$z_i^t \ge \sum_{k\in K} y_{ik}^t - \sum_{k\in K} y_{ik}^{t-1}, \qquad \forall i\in\hat{\mathcal{P}}, t\in T, y_{ik}^{-1}=0 \tag{27}$$

$$w_{ik} \ge y_{ik}^t \qquad \forall i\in\hat{\mathcal{P}}, t\in T, k\in K \tag{28}$$

Objective function (19) minimizes the waiting times and the total number of linacs assigned to a patient during his/her treatment. Constraints (20) ensure each patient is scheduled on at most 1 linac per day. Constraints (21) prevent assigning patients to non-eligible linacs. Constraints (22) limit the number of sessions delivered to each patient. Constraints (23) ensure daily treatments. Constraints (24) and (25) make sure all patients are treated before their due days but after their release days. Constraints (26) ensure the capacity of each linac is not exceeded where $\hat{C}_k^t$ is the available capacity of linac k, day t after deducting the treatments of fixed patients. Constraints (27) define the first days of treatment. Constraints (28) keep track of a patient being assigned to a linac during his/her course of treatment.

3.2 Phase 2: ordering patients on linacs - IP model

Given the assignment of patients to days and linacs, let P_k^t be the set of patients scheduled on linac k, day t. P_k^t is composed of two subsets $\bar{P}_k^t$, including fixed patients from the previous period and $\hat{P}_k^t$, including new patients. The linac assigned to patient i on day t is known from the first phase and denoted as k_i^t.

The variables used in the model are similar to the variables used in the first model for the whole problem.

- x_{is}^{tk}: binary variables indicating that patient i is assigned to slot s.
- $\bar{\Delta}_i^t$ and $\underline{\Delta}_i^t$: representing the deviation from the preference time window of non-palliative patient $i \in \mathcal{P}^N$ on day t (sooner or later).
- $\bar{\theta}_i$ and $\underline{\theta}_i$: representing patient's earliest and latest appointment time slot.
- $\bar{\phi}_i^t$ and $\underline{\phi}_i^t$: representing changes in appointment times (to sooner and later) of fixed patient $i \in \bar{\mathcal{P}}$ on day t.

Model:

$$\text{minimize } \omega_3 \sum_{i\in\mathcal{P}^N, t\in T} (\bar{\Delta}_{it} + \underline{\Delta}_{it}) + \omega_4 \sum_{i\in\mathcal{P}^N} (\bar{\theta}_i - \underline{\theta}_i) + \omega_5 \sum_{i\in\bar{\mathcal{P}}, t\in T} (\bar{\phi}_i^t + \underline{\phi}_i^t) \tag{29}$$

subject to

$$\sum_{s\in S} x_{is}^{tk} = 1 \qquad \forall t \in T, k \in K, i \in P_k^t \tag{30}$$

$$\sum_{i\in P_k^t} x_{is}^{tk} \leq 1 \qquad \forall t \in T, k \in K, s \in S \tag{31}$$

$$x_{is}^{tk} \leq 1 - x_{i's'}^{tk}, \qquad \forall t \in T, k \in \mathcal{K}, i, i' \in P_k^t, i \neq i' \quad s = 0, \ldots, |\mathcal{S}| - p_i \quad s' = s+1, \ldots, s+p_i-1 \tag{32}$$

$$x_{is}^{tk} = 0, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, i \in \mathcal{P}_k^t, \quad s = |\mathcal{S}| - p_i + 1, \ldots |\mathcal{S}-1| \tag{33}$$

$$\bar{\Delta}_i^t \geq x_{is}^{tk}(t_i^{min} - s), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}_k^{Nt}, s \in \mathcal{S} \tag{34}$$

$$\underline{\Delta}_i^t \geq x_{is}^{tk}(s - t_i^{max}), \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \hat{\mathcal{P}}_k^{Nt}, s \in \mathcal{S} \tag{35}$$

$$\bar{\theta}_i \geq \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^{Nt}, k = k_i^t \tag{36}$$

$$\underline{\theta}_i \leq \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, i \in \hat{\mathcal{P}}^{Nt}, k = k_i^t \tag{37}$$

$$\bar{\phi}_i^t \geq \bar{s}_i^t - \sum_{s\in\mathcal{S}} s x_{is}^{tk}, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}}_k^t \tag{38}$$

$$\underline{\phi}_i^t \geq \sum_{s\in\mathcal{S}} s x_{is}^{tk} - \bar{s}_i^t, \qquad \forall t \in \mathcal{T}, k \in \mathcal{K}, \forall i \in \bar{\mathcal{P}}_k^t \tag{39}$$

Objective function (29) minimizes the deviation from preference time windows, inconsistency in appointment times and changes of appointments of fixed patients with the respective weights ω_3, ω_4 and ω_5. Constraints (30) make sure each patient is assigned to exactly one slot in their treatment day. Constraints

(31) make sure each slot is assigned to at most one patient. Constraints (32) ensure the number of required time slots for each patient by blocking the next slots on the linacs. Constraints (33) ensure no patients are scheduled on the last slot(s) of the days which are not sufficient for their treatment time. Constraints (34) and (35) define the deviation from the time preferences. Constraints (36) and (37) define the earliest and latest appointment time of a patient during his/her treatment course. Constraints (38) and (39) define the changes in appointment times of fixed patients.

3.3 Phase 2: order of patients on linacs - CP model

As an alternative for the MILP model, a Constraint Programming model is proposed for phase 2. The model uses variables and constraints supported by IBM CP Optimizer [7]. CP Optimizer supports two main types of decision variables, namely *integer* and *interval variables*. An integer variable is a discrete integer while an interval variable consists of a start point and an end point, representing an interval of time. Another type of variable used in this model is a *sequence variable*. A sequence variable consists of a set of interval variables.

In this model, the parameters are similar to the ones in the MILP models. We hereby introduce the variables and constraints.

- The interval variable y_i^t represents the appointment duration of patient i on day t. The duration (d) of variable y_i^t is defined by the start (s) and the end (e) of the fraction of the corresponding patient. Recall that C_k^t is the capacity of linac k in day t.

$$\forall t \in \mathcal{T}, \forall k \in \mathcal{K}, i \in \mathcal{P}_k^t : \begin{cases} s(y_{ki}^t) \in [0, C_k^t - p_i] \\ e(y_{ki}^t) \in [C_k^t - p_i, C_k^t] \\ d(y_{ki}^t) = p_i \end{cases} \tag{40}$$

- The interval variable x_i represents the time span of appointment times of patient i. x_i starts at the earliest appointment time and ends at the latest appointment end time of patient i during his/her treatment period. The longer the duration of x_i, the more inconsistency appointment times of the corresponding patient during the treatment. This is imposed by the specialized constraint *span* of CPOptimizer.

$$i \in \mathcal{P}_k^t : \begin{cases} s(x_i) \in [0, C_k^t] \\ e(x_i) \in [0, C_k^t] \\ span(x_i, [y_{ki}^t | \forall t \in \mathcal{T}, \forall k \in \mathcal{K}]) \end{cases} \tag{41}$$

- The *sequence variable* seq_k^t contains all interval variables y_{ki}^t corresponding to all fractions performed on linac k on day t. The specialized constraint *noOverlap* is imposed on seq_k^t to make sure that a fraction must end before the next one starts.

$$\forall t \in \mathcal{T}, k \in \mathcal{K} : \begin{cases} seq_k^t = \{y_{ki}^t | i \in P_k^t\} \\ \text{noOverlap}(seq_k^t) \end{cases} \tag{42}$$

- $\bar{\Delta}_i^t$ and $\underline{\Delta}_i^t$ represent the deviation (to sooner or later) from the preference time windows of new, non-palliative patients. They are defined by constraints (44).

$$\forall t \in \mathcal{T}, i \in \hat{\mathcal{P}} : \begin{cases} \bar{\Delta}_i^t \in [0, C_k^t] \\ \underline{\Delta}_i^t \in [0, C_k^t] \end{cases} \tag{43}$$

$$\forall t \in \mathcal{T}, k \in \mathcal{K}, i \in \hat{\mathcal{P}} : \begin{cases} \bar{\Delta}_i^t \geq t_i^{min} - StartOf(y_{ki}^t) \\ \underline{\Delta}_i^t \geq StartOf(y_{ki}^t) - t_i^{max} \end{cases} \tag{44}$$

The objective function (45) consists of three parts, each of which minimizes the following objectives respectively: (1) the violation of time window preference of patients; (2) in-consistency in appointment times; and (3) changes in appointments of fixed patients.

$$\begin{aligned} \text{minimize } & \omega_3 \sum_{t \in \mathcal{T}} \sum_{i \in P_k^t} (\bar{\Delta}_i^t + \underline{\Delta}_i^t) \\ & + \omega_4 \sum_{i \in \mathcal{P}^N} LengthOf(x_i) \\ & + \omega_5 \sum_{i \in \bar{\mathcal{P}}_k^t} abs(StartOf(y_{ki}^t) - \bar{s}_i^t) \end{aligned} \tag{45}$$

Category	Percentages	Deadline (days)
P1	0.44%	1
P2	27.14%	3
P3	41.36%	14
P4	31.06%	28

Table 1: Distribution of patient categories

4 Experiment

4.1 Data description

In this section, we present an overview of the data from CHUM and how we generate test instances based on real data. CHUM is a large cancer center in Montréal, Canada. In 2018, CHUM has around 4400 consultations, 3500 new patients and 40.000 treatments. Currently, the scheduling is made manually. The center is equipped with 11 linacs, divided into 4 groups. Linacs from each group are almost identical and patients can switch between them during treatments. The scheduling is conducted sequentially, i.e. the appointments are made once the treatment plan of a new patient is decided. Usually, the next week is full with some slots for emergency treatments, the second week is about 70% full while the third week has a couple of booked slots. Preferences in appointment times only apply for some patients which live far away from the center. Based on the diagnosis of the patients, the doctors will decide their treatment plans. Currently, there are 356 treatment plans. Each treatment plan is associated with a priority level and one or two groups of eligible linacs. There are 4 types of priority, from P1 to P4. Patients with P1 and P2 priority are palliative ones which require urgent treatment. Patients with P2 and P3 priorities are curative ones which have 14 and 28 days of due day deadline respectively. The percentages of patients regarding categories and the deadlines are listed in table 1. The number of sections and fraction lengths depend on both the treatment plan and the specific condition of the patient. However, each treatment plan usually has a small range of fraction length values. All fraction length values are multiples of 5 minutes.

4.2 Data generation

We generate test data based on the distribution of real data from CHUM. Data is generated using the following steps:

1. A pool of patients is generated. Each patient has a treatment plan, which also determines his/her priority, number of fractions, fraction length and eligible linac groups. This information is generated based on the distribution of the real data.
2. An instance size is determines by the number of linacs (can be limited in both the number of linacs in each group and the number of groups), the planning horizon (number of days that the schedule should be made), and the number of new patients.
3. Given an instance, fixed patients are assigned so that the occupation rate of each week in line with real data. To do so, non-palliative patients are iteratively taken from the patient pool and assigned to a series of free slots in the schedule, until the desired occupation rates are met (a series is a set of fractions with identical appointment time in a couple of consecutive days).
4. Finally, a pre-defined number of patients is taken from the patient pool as the new patients for the instance. The release date of each patient is generated randomly. The due day of each patient is then calculated according his/her release date and urgency level. Currently, to simplify the problem, we generate release dates of patients so that their due days are always within the planning horizon.

4.3 Numerical results

All experiments are carried out on an Intel Core i7-4600 2.90GHz * 4 machine with 8GB of RAM, running Ubuntu 16.04 64-bit. IBM ILOG CPLEX and CP Optimizer version 12.8.0 are used as the solvers.

The test set consists of 16 instances with a horizon of 21 days. The numbers of linacs range from 1 to 7, divided into up to 3 groups. The numbers of patients range from 10 to 80 patients and are adjusted according to the number of linacs so that the instances are feasible. Phase 1 is given 10 seconds of time budget while

					Phase 1			Phase 2 MILP				Phase 2 CP			
Instance	K1	K2	K3	P	obj	gap (%)	time	obj	time	gap (%)	bound	obj	time	gap (%)	bound
ins01	1	0	0	10	31	0	0.06	**0**	6.34	0	0.0	**0**	1.17	0	0
ins02	1	0	0	25	35	0	0.05	7665	30.26	128	-2164.0	**25**	30.13	12	8
ins03	2	0	0	20	23	0	0.23	**0**	9.82	0	0.0	**0**	2.02	0	0
ins04	2	0	0	30	36	0	0.27	10714	30.20	126	-2743.0	**0**	16.19	0	0
ins05	2	0	0	40	47	0	0.49	12325	30.26	132	-3891.0	**0**	3.95	0	0
ins06	2	1	0	40	3571	0	0.64	12861	30.27	121	-2720.0	**0**	5.76	0	0
ins07	2	1	0	50	55	0	0.63	12724	30.37	133	-4304.0	**69**	30.23	19	0
ins08	2	1	0	60	271	0	0.62	12752	30.39	133	-4266.0	**650**	30.30	81	2
ins09	3	0	0	40	40	0	1.92	459	30.28	100	0.0	**0**	17.93	0	0
ins10	3	0	0	50	58	0	2.53	17946	30.25	126	-4692.0	**0**	28.10	0	0
ins11	3	0	0	60	60	0	8.15	13077	30.45	146	-5955.0	**429**	30.27	63	0
ins12	3	1	1	50	110	0	0.87	24	30.08	100	0.0	**0**	15.32	0	0
ins13	3	1	1	60	60	0	1.67	16881	30.66	128	-4691.0	**186**	30.25	35	0
ins14	3	1	1	70	131	0	1.46	25153	30.48	122	-5454.0	**49**	30.37	14	0
ins15	4	2	1	70	3473	0	86.90	21126	30.67	123	-4764.0	**327**	30.37	59	0
ins16	4	2	1	80	138	0	3.26	21738	30.83	126	-5669.0	**0**	18.02	0	0

Table 2: Results with 30 seconds time limit

in phase 2, both MILP and CP are given 30 seconds of time limit. The results can be found in table 2. The first column shows instance names, the next three columns display number of linacs of each group, the fifth column is number of patients. For each model, the objective values, optimality gaps, solving times and best bounds are reported.

It can be seen from the results that phase 1 is easy to solve. Optimal solutions are achieved in all instances within less than 8 seconds, with the exception of 95 seconds in instance *ins15*. In phase 2, CP tremendously outperforms MILP in all instances. Given only 30 seconds of run time, CP Optimizer achieves optimal solutions in 9 out of 16 instances. In the remaining 7 instances, good objective values are also obtained with 12% to 81% gap from the lower bounds. By contrast, CPLEX does not perform very well in most instances, except *ins01* and *ins03* where optimal solutions are obtained. The remaining instances in general have bad objective values compared to results from CP. Especially, the lower bounds offered by CPLEX are either negative or 0, which results in large optimality gaps. This can be explained by the appointment consistency terms in the objective function of the MILP model.

In table 3, the results with a larger time limit (1200 seconds) are reported. With the large time limit, CP Optimizer achieves optimal solutions in all instances while CPLEX closes the gaps in 9 out of 16 instances. In the remaining 7 instances, CPLEX also obtains good solutions compared to the results with 30 seconds, but the optimality gaps are still large. We come to the conclusion that CP performs very well in the RTSP while MILP might benefit from another formulation which can offer better bounds.

5 Conclusions and future work

In this paper, we have introduced a 2-phase approach for the Radiotherapy Scheduling Problem. A MILP model is proposed for phase 1 while a MILP and a CP model are proposed for phase 2. We also present how realistic instances are generated based on real instances from a cancer center in Montréal. The numerical results show that CP outperforms MILP in all instances and is able to solve realistic instance sizes.

As for future work, we aim to solve large-sized instances with more sophisticated algorithms. It is also interesting to take into account cancellations and future changes in the schedule.

Acknowledgements This work is funded by the FWO (Research Foundation Flanders) grant for a long stay abroad (V413919N) and YouReCa - Junior Mobility Programme (JUMO/13/031). We would like to thank Prof. Greet Vanden Berghe and the two anonymous reviewers for their suggestions and comments.

					Phase 2 MILP				Phase 2 CP			
Instance	K1	K2	K3	P	obj	time	gap (%)	bound	obj	time	gap (%)	bound
ins01	1	0	0	10	**0**	5.89	0	0.0	**0**	0.99	0	0
ins02	1	0	0	25	441	1200.11	100	1.6	**19**	233.04	0	19
ins03	2	0	0	20	**0**	9.47	0	0.0	**0**	2.02	0	0
ins04	2	0	0	30	**0**	183.76	0	0.0	**0**	16.10	0	0
ins05	2	0	0	40	344	1200.38	100	0.0	**0**	3.97	0	0
ins06	2	1	0	40	**0**	37.72	0	0.0	**0**	5.77	0	0
ins07	2	1	0	50	973	1200.15	100	0.3	**3**	286.15	0	3
ins08	2	1	0	60	743	1200.13	100	0.2	**0**	54.73	0	4
ins09	3	0	0	40	**0**	210.25	0	0.0	**0**	18.07	0	0
ins10	3	0	0	50	42	1200.14	100	0.0	**0**	28.36	0	0
ins11	3	0	0	60	295	1200.42	100	0.0	**0**	49.79	0	0
ins12	3	1	1	50	**0**	52.44	0	0.0	**0**	15.24	0	0
ins13	3	1	1	60	7	1200.33	100	0.0	**0**	45.26	0	0
ins14	3	1	1	70	**0**	313.13	0	0.0	**0**	35.94	0	0
ins15	4	2	1	70	**29**	802.17	0	29.0	32	1200.30	10	3
ins16	4	2	1	80	**0**	280.61	0	0.0	**0**	18.12	0	0

Table 3: Results with 1200 seconds time limit

References

1. Canadian cancer statistics, a 2018 special report. http://www.cancer.ca/en/cancer-information/cancer-101/canadian-cancer-statistics-publication/?region=on, 2018. [Online; accessed 08-August-2019].
2. Z. Chen, W. King, R. Pearcey, M. Kerba, and W. J. Mackillop. The relationship between waiting time for radiotherapy and clinical outcomes: a systematic review of the literature. *Radiotherapy and Oncology*, 87(1):3–16, 2008.
3. C. Coles, L. Burgess, and L. Tan. An audit of delays before and during radical radiotherapy for cervical cancer–effect on tumour cure probability. *Clinical oncology*, 15(2):47–54, 2003.
4. D. Conforti, F. Guerriero, and R. Guido. Optimization models for radiotherapy patient scheduling. *4OR*, 6(3):263–278, 2008.
5. D. Conforti, F. Guerriero, and R. Guido. Non-block scheduling with priority for radiotherapy treatments. *European Journal of Operational Research*, 201(1):289–296, 2010.
6. T. Kapamara and D. Petrovic. A heuristics and steepest hill climbing method to scheduling radiotherapy patients. In *Proceedings of the 35th International Conference on Operational Research Applied to Health Services, Catholic University of Leuven, Belgium*. Citeseer, 2009.
7. P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.
8. W. J. Mackillop. Killing time: the consequences of delays in radiotherapy. *Radiotherapy and Oncology 84.1*, 1(4), 2007.
9. J. Maschler and G. R. Raidl. Particle therapy patient scheduling with limited starting time variations of daily treatments. *International Transactions in Operational Research*, 2018.
10. J. Maschler, M. Riedler, M. Stock, and G. R. Raidl. Particle therapy patient scheduling: First heuristic approaches. In *Proceedings of the 11th Int. Conference on the Practice and Theory of Automated Timetabling. Udine, Italy*, 2016.
11. S. Petrovic and P. Leite-Rocha. Constructive approaches to radiotherapy scheduling. In *Proceedings of the World Congress on Engineering and Computer Science*, pages 722–727, 2008.
12. S. Petrovic, W. Leung, X. Song, and S. Sundar. Algorithms for radiotherapy treatment booking. In *25th Workshop of the UK planning and scheduling special interest group*, pages 105–112, 2006.
13. A. Sauré, M. A. Begen, and J. Patrick. Dynamic multi-priority, multi-class patient scheduling with stochastic service times. *European Journal of Operational Research*, 280(1):254–265, 2020.
14. A. Saure, J. Patrick, S. Tyldesley, and M. L. Puterman. Dynamic multi-appointment patient scheduling for radiation therapy. *European Journal of Operational Research*, 223(2):573–584, 2012.
15. B. Vieira, D. Demirtas, J. B. van de Kamer, W. Jeroen, L.-M. Rousseau, N. Lahrichi, and W. van Harten. A mathematical programming model for radiotherapy scheduling with time windows. In *Springer Proceedings of the Health Care Systems Engineering conference, 30 May-01 June, Montréal, Canada*, 2019.
16. P. Vogl, R. Braune, and K. F. Doerner. Scheduling recurring radiotherapy appointments in an ion beam facility. *Journal of Scheduling*, 22(2):137–154, 2019.

MISTA 2019

Lot Scheduling Problems on Identical Parallel Machines

Biber Nurit · Mor Baruch · Shapira Dana

Abstract We consider the problem of lot scheduling on m parallel identical machines. Each lot may contain one or more orders, in which the orders may be of different sizes. In case the remaining portion of the lot is less than the size of the order, the order is split between consecutive lots. In this paper the goal is to minimize the makespan on identical parallel machines in case of lot scheduling. We prove that the makespan can be solved in pseudo-polynomial time by introducing a Dynamic Programming algorithm solution for $m \geq 2$, and propose an approximation algorithm with constant worst case performance ratio of 2.

1 Introduction

Scheduling Theory is a fundamental branch of Operations Research, which is described by Pinedo [13] as a form of decision-making that plays a crucial role in manufacturing and service industries. Within scheduling theory, *Lot-Scheduling* with order-splitting, is an emerging research class, in which the producer receives the customers' orders of different sizes which are processed in lots, such that the total size of the orders processed in a single lot cannot exceed its capacity.

We consider the case of lot scheduling on identical parallel machines, where the scheduler may process multiple orders in a single lot. The current paper considers lot scheduling with *order splitting*, i.e. an order that cannot be completed in the current active lot must be completed in the consecutive lot. Lot scheduling problems consider the following realistic features: (i) all lots have a predefined identical capacity; (ii) all lots have identical processing times; (iii) the size of a single order cannot exceed

Biber Nurit
Ariel University
E-mail: nurits.nurit@gmail.com

Mor Baruch
Ariel University
E-mail: baruchm@g.ariel.ac.il

Shapira Dana
Ariel University
E-mail: shapird@g.ariel.ac.il

the common size (capacity) of a lot; and (iv) an order can be split only between two consecutive lots. The completion time of a particular order, in case it was not split, is the completion time of the lot in which it was completed in, and the weighted completion time, otherwise (the formal definition is given later).

Lot processing on parallel machines is motivated by its commonness in advanced industry, which allows manufacturing in parallel, thus enhancing the production by saving expenses and time. As example, during the manufacturing process of integrated circuits (IC), electronic tests are performed under elevated temperature and other stress conditions, to ensure the required high quality and the reliability of the produced semiconductors before shipping them to the final users. Another example for lot scheduling is the production of glass, where in initial stages the raw materials is housed in large silos for several days. In general, lot-scheduling problems are very common in productions that must be kept raw or for materials that are handled in certain heating or cooling containers for a period of time to form the end product.

In this paper we focus on minimizing the makespan in case of lot scheduling with order splitting on identical parallel machines. Using the three field notation for scheduling problems proposed by Graham et al. [7], the problem is denoted by $P_m|lot, split|C_{\max}$, where $C_{\max}$ denotes the maximal completion time. We prove that this problem can be solved in pseudo-polynomial time by introducing a *Dynamic Programming* (DP) algorithm solution.

In classical scheduling theory, when lot scheduling is not considered, minimizing the makespan on parallel machines, i.e., $P_m||C_{\max}$ is known to be NP-Hard in the ordinary sense. Therefore, it has been solved by several methods such as DP, branch and bound, and integer programming. These methods provide optimal solutions for reasonable sized problems.

Ronald Graham [6] concentrates on the case that the jobs are selected in *Longest Processing Time* (LPT) order for minimizing the makespace on m parallel machines, and shows that LPT is within a factor of $\frac{4}{3}-\frac{1}{3m}$ to optimum. In another paper, Ghalami et al. [5] design a parallel approximation DP algorithm for the problem of scheduling jobs on parallel identical machines to minimize the makespan. To obtain parallelism the dependencies between the sub-problems in the DP formulation are modelled as a directed graph. Potts [14] presents a heuristic method for minimizing the makespan on unrelated parallel machines. The heuristic paradigm uses linear programming and an enumerative method. For two machines, it is shown that the heuristic has a worst case performance ratio of $\frac{3}{2}$, and for three or more machines the worst case performance ratio is 2.

Li et al. [10] consider the problem of scheduling jobs on parallel identical machines to minimize the makespan, and propose a parallel approximation algorithm which is designed and optimized for running on *Graphics Processing Units* - GPUs. The algorithm is a PTAS, based on a higher-dimensional DP approach, where dimensionality refers to the number of cores, which are the variables in the dynamic programming equation characterizing the problem.

Wang et al. [19] consider the identical parallel machine scheduling problem to minimize simultaneously two objectives: the *Total Energy Consumption* (TEC) and the makespan. The authors propose a constructive heuristic method with a local search strategy that is suitable for medium- and large-scale instances. Oron et al. [11] consider the problem of scheduling on identical parallel machines with *controllable* processing times, to minimize the makespan, meaning that the duration for each job is determined through the allocation of the resources according to the job's operation. They prove

that the problem is NP-hard based on a reduction to the *Weighted Workload Partition* problem, which itself reduces to the well-known 3-Partition problem, and provide an optimal resource allocation and FPTAS algorithm to several underling variants.

The present status and possible potential solutions for $P_m||C_{\max}$ are given in Blazewicz et al.[1], who compiled a systematic consistent survey of mathematical programming formulations for solving various machine scheduling problems. Cheng and Sin [2] also conduct a survey composed of solutions to machine scheduling problems, and focus on the identical parallel machines environment.

A related class to lot scheduling is *batch scheduling.* Batch scheduling is defined as follows. A machine may be able to process a batch of up to a predefined number, k, of jobs at the same time. The processing times of the jobs in a batch may not be all the same, and the entire batch is finished only when the last job of the batch has been completed, implying that the completion time of the entire batch is dominated by the job with the longest processing time. If $k = 1$, then the problem reduces to a conventional scheduling environment. Another special case that is of interest is $k = \infty$, i.e., there is no limit on the number of jobs the machine can handle at the same time. The main difference between lot and batch scheduling refers to the number of jobs that may be handled at once. In lot scheduling, the number of orders scheduled in a given lot is limited by the lot capacity, whereas in batch scheduling the batch can process a constant number of jobs simultaneously, independent of the jobs' processing times.

There is extensive literature on models that integrate scheduling with batching decisions. Jobs may be batched if they share the same setup on a machine or when a machine can process several jobs simultaneously. Potts et al. [15] provide an extensive review on scheduling with batching including DP algorithms for solving these type of problems.

Parallel-batch scheduling was first introduced by Lee et al. [9,17,18] as part of a semiconductor manufacturing environment. Uzsoy et al. review models related to performance evaluation and production planning. They propose efficient DP-based algorithms for minimizing a number of performance measures on a single batch processing machine, present heuristics for a number of parallel identical batch processing machines problems and provide worst case error bounds. Ozturk et al. [12] present a branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize the makespan, where there are identical machines with a fixed capacity and the number of jobs in a batch cannot exceed the machine capacity.

Single machine lot scheduling with order splitting to minimize the makespan without preemption, i.e. $1|lot, split|C_{\max}$, is trivial, as any schedule obtains the same number of lots. Although lot scheduling is a very common production style, we are only aware of two papers that investigate this area of research. Hou et al. [8] study lot scheduling on a single machine and show that minimizing the total completion time, i.e., $1|lot, split|\sum C_j$, in the three field notation, can be achieved when orders are sequenced in *Shortest Processing Time first* order, or SPT for short, and arranged sequentially on lots. Zang et al. [20] generalize the objective function to total weighted completion time on the same setting, i.e., $1|lot, split|\sum w_j C_j$, and prove that an optimal schedule is given by a *Weighted Shortest Processing Time first* order, WSPT, where w_j is the weight of order j.

For the non lot version, scheduling problems on identical parallel machines in order to minimize the makespan has been proved to be an NP-hard problem [3,16]. Ghalami et al. [4] present a parallel approximation algorithm for the same problem setting, $P_m||C_{\max}$, where their solution is specifically designed for execution on shared-memory

parallel machines. The authors provide experimental analysis on data generated from realistic probability distributions illustrating the efficiency of the proposed parallel approximation algorithm. Being that lot scheduling is a more complex setting, obviously $P_m|lot, split|C_{\max}$ is also NP-hard. In this paper we present a DP solution for minimizing the makespan on two machines, and propose an approximation algorithm with constant worst case performance ratio of 2 for the generalized version of any finite number m of parallel machines, i.e., $Pm|lot, split|C_{\max}$. The proposed heuristic is shown to approximate the optimal makespan within a constant factor, for any arbitrary order arrangement.

The paper is constructed as follows. Section 2 gives a formal description of the problem, and includes the involved notations. Section 3 presents the DP solution for two parallel identical machines, $P_2|lot, split|C_{\max}$, and Section 4 generalizes it to $m \geq 2$ parallel identical machines, $P_m|lot, split|C_{\max}$. In Section 4 we propose a polynomial time heuristic for the general problem, and Section 5 concludes.

2 Formal Problem Description

The models considered in this paper fall in the category of *offline scheduling* problems, which implies that all the problem's data (lot processing times, lot capacity, etc...) are known in advance, and can be taken into account in the optimization process.

There are n given orders to be processed on m parallel identical machines with the lot processing settings. All lots have an identical capacity L and processing time P and are executed continuously on the machines. Each machine can handle a single lot at a time. The orders may have different sizes, which are limited by the lot capacity. Several orders may be scheduled on the same lot as long as their total size (total processing times) is less than or equal to L. When an order cannot be processed entirely in the current lot since its size exceeds the remaining capacity, the order can be split into two consecutive lots, the current and the following one.

The completion time of the orders are calculated as follows: All orders scheduled on the same lot have a joint completion time, which is the completion time of that lot. When an order is split over two (consecutive) lots, each part of the order may be delivered to the customer immediately the lot is finished, so the completion time is the sum of the percentage of the order in each lot, multiplied by the lot completion time. For example, if an order is split, processing 35% of that order in lot k, and the remaining part of the order, 65%, in lot $k+1$, then the order's completion time is $0.35 \cdot k \cdot P + 0.65 \cdot (k+1) \cdot P$.

The processing time of each order, referred to as the size of the order, is denoted by $o_k > 0$, $0 < k \leq n$. The orders should be arranged optimally in lots, given the parameters of the lot's capacity and the lot's processing time, L and P, respectively. Let o_{max} denote the maximum size among all orders, i.e., $o_{max} = \max_{1 \leq k \leq n} o_k$. Referring to the requirement that an order that has been split should be processed in two consecutive lots, implies that the size of every order cannot be greater than the lot capacity, that is, $o_{max} \leq L$.

As mentioned above, in lot scheduling problems, the orders' completion time is defined by the completion time of the lot it is completed in, that is, by multiplying the lot processing time by the number of lots processed so far. Thus, the completion time of order k is given by $\left\lceil \frac{\sum_{j=1}^{k} o_j}{L} \right\rceil P$. The machine setting in this work is more involved,

and includes more than a single machine. For machine k, named M_k, $\sum_{j \in M_k} o_j$ is the sum of sizes of all orders scheduled on M_k. In the lot production setting, the makespan, $C_{\max}$, is calculated according to the completion time of the last order processed on any of the individual machines. We use S_k to denote the sum of the first k order sizes, independently of the machine they are processed on, i.e., $S_k = \sum_{j=1}^{k} o_j$. In particular, S_n denotes the entire sum of all n order sizes, $S_n = \sum_{j=1}^{n} o_j$.

3 Minimizing Makespan with Lot Scheduling

In this section we consider lot scheduling with splits, and propose DP solutions to the case of a fixed number of parallel identical machines. We start with minimizing the makespan in the setting of lot scheduling on two parallel identical machines, $P_2|lot, split|C_{\max}$, and generalize it to any fixed number, m, of $m \geq 2$ parallel identical machines, $P_m|lot, split|C_{\max}$. Pinedo [13] shows that $P2||C_{\max}$ is ordinary NP-hard by a reduction from the well known 2-partition problem (2PAR). In 2PAR one needs to determine whether a given set of positive integers S can be partitioned into two subsets S_1 and S_2 such that the sum of numbers in both sets is equal. As the classic problem $P_2||C_{\max}$ is already NP-hard, the extended problem to lot scheduling, $P_2|lot, split|C_{\max}$, is obviously also NP-hard. We present pseudo-polynomial DP optimal solutions, showing that even the generalized problem remains NP-hard in the ordinary sense.

3.1 Optimal Makespan on Two Identical Parallel Machines

We consider two machines, M_1 and M_2. Given a set of n orders, for each order $1 \leq j \leq n$ and each sum of order sizes t, $0 \leq t \leq S_n$, $f(j,t)$ is defined as the minimum completion time of subset $\{1, ..., j\}$ of orders, scheduled on two identical parallel machines, arranged in lots, having total size t on M_1. Note that the recursion is not symmetrical on both machines. The computation of $f(j,t)$ is accomplished by constructing a table of size $(n+1) \times (S_n + 1)$. Each state (j,t) of the DP maintains two variables, $O^k_{(j,t)}$, $k = 1, 2$, according to the minimal scheduled achieved so far. Each variable is used to reflect the sum of the sizes of all orders $\{1, \ldots, j\}$ scheduled on the k^{th} machine, M_k, in which t is the maximal total sum of order sizes scheduled by the optimal schedule so far on M_1. More precisely, $O^k_{(j,t)}$ is defined by the following recursions:

$$O^1_{(j,t)} = \begin{cases} \infty, & t < o_j \\ O^1_{(j-1,t-o_j)} + o_j, & t \geq o_j \end{cases} \qquad O^2_{(j,t)} = \left\{ O^2_{(j-1,t)} + o_j \right. \tag{1}$$

The initialization is by $O^k_{(0,0)} = 0$ and $O^k_{(0,t)} = \infty, \quad \forall t : 0 < t \leq S_n$. As the definition of $f(j,t)$ is based on $O^k_{(j,t)}$, the unsymmetrical treatment of $O^k_{(j,t)}$ for both machines, is inherited. The recursion function $f(j,t)$ is computed based on the minimum makespan for lot production attained for the subset $\{1, ..., j-1\}$ of orders having total order sizes less or equal to t on machine M_1. The boundary conditions are as follows:

1. $f(0,0) = 0$,
2. $f(j,0) = \left\lceil \frac{S_j}{L} \right\rceil P, \qquad 0 < j \leq n$

3. $f(0,t) = \infty, \qquad 0 < t \leq S_n$

Condition 2 is the result of lot-scheduling all orders $\{1, \cdots, n\}$ on machine 2. Condition 3 represents the empty set of orders.

The recursion step of $f(j,t)$ is:

$$f(j,t) = \begin{cases} \max(f(j-1,t), \left\lceil \frac{O^2_{(j,t)}}{L} \right\rceil P) & , t < o_j \\ \min\Big(\max(f(j-1,t-o_j), \left\lceil \frac{O^1_{(j,t-o_j)}}{L} \right\rceil P), & \\ \quad \max(f(j-1,t), \left\lceil \frac{O^2_{(j,t)}}{L} \right\rceil P) \Big) & , t \geq o_j \end{cases} \tag{2}$$

The first line reflects the case where order j must be scheduled on machine 2 since t is less than o_j, otherwise order j may be scheduled on either Machine 1 or Machine 2, presented on the last two lines. The first and second arguments of the minimum function are the $C_{\max}$ values when order j is scheduled on Machine 1 or Machine 2, respectively. The optimal solution is finally given by $\min\limits_{0 \leq t \leq S_n} f(n,t)$.

Theorem 1 *The optimal makespan on two identical parallel machines can be solved in $\mathcal{O}(nS_n)$ processing time.*

Proof The optimal makespan on two identical parallel machines is computed by the DP based on Equation 2. The recursive function is calculated for every job j, $0 \leq j \leq n$, and for every size t, $0 \leq t \leq S_n$, using constant time for each cell. Reconstructing the solution is done by finding the minimum cost obtained in the final row $f(n,t)$, for $0 \leq t \leq S_n$, which uses additional $\mathcal{O}(S_n)$ time, followed by backtracking to $f(0,0)$ for an addition of $\mathcal{O}(n + S_n)$ processing time. In conclusion, the total processing time is $\mathcal{O}(nS_n)$. ∎

Consider, for example, the following setting: the lot's capacity and processing time are $L = 3$ and $P = 20$, respectively, and five order sizes $o_j = \{1, 2, 2, 2, 3\}$. The minimal cost is given by $f(5,5) = 40$, implying that an optimal arrangement is scheduling orders 1, 2 and 3 on Machine 1 and orders 4, 5 on machine 2, which is depicted in Figure 1.

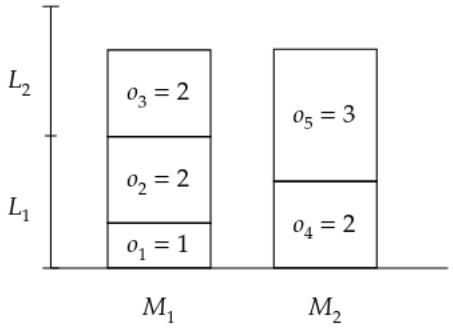


Fig. 1: Lot-scheduling arrangement for L=3, P=20 and o_j={1,2,2,2,3}.
L_1 and L_2 represent the two lots used on each machine.

3.2 Optimal Makespan on $m \geq 2$ Identical Parallel Machines

In this subsection the solution for minimizing the makespan on two identical parallel machines is extended to solving $P_m|lot, split|C_{\max}$. The dynamic programming table is of m dimensions, $(n+1) \times (S_n+1) \times \cdots \times (S_n+1)$, where n is the number of orders. For each order $1 \leq i \leq n$ and each total order sizes $t_1, \cdots, t_{m-1}$, where $0 \leq t_1, \cdots, t_{m-1} \leq S_n + 1$, the cell $T[j, t_1, \ldots, t_{m-1}]$ is defined as the minimum completion time of the subset $\{1, \ldots, j\}$ of orders, scheduled on m identical parallel machines arranged in lots, having total order sizes t_k on machine M_k, $1 \leq k < m$. Machine m, M_m, does not have a corresponding index, being that it is the default machine, that is, if order i is not scheduled on any one of the previous machines, $M_1, \ldots, M_{m-1}$, order i is scheduled automatically on M_m.

The computation of $f(j, t_1, \ldots, t_{m-1})$ is accomplished by maintaining m variables $O^k_{(j,t_1,\ldots,t_{m-1})}$, for $k = 1, \ldots, m$, which are used to reflect the sum of sizes of the subset of orders from the set of orders $\{1, \ldots, j\}$ scheduled on Machine k, in which t_k is the total size on Machine k, M_k, given only for $k \leq m-1$. More precisely, $O^k_{(j,t_1,\ldots,t_k,\ldots,t_{m-1})}$ is defined by the following recursion.

For $1 \leq k < m$, $\quad O^k_{(j,t_1,\ldots,t_k,\ldots,t_{m-1})} = \begin{cases} \infty, & t_k < o_j \\ O^k_{(j-1,t_1,\ldots,t_k-o_j,\ldots,t_{m-1})} + o_j, & t_k \geq o_j \end{cases}$

For $k = m$, $\quad O^m_{(j,t_1,\ldots,t_{m-1})} = \left\{ O^m_{(j-1,t_1,\ldots,t_{m-1})} + o_j \right.$, where $O^k_{(0,\ldots,0)} = 0$ and $O^k_{(0,t_1,\ldots,t_k,\ldots,t_{m-1})} = \infty$ in case $0 < t_k \leq S_n$.

The recursion function $f(j, t_1, \ldots, t_k, \ldots, t_{m-1})$ is computed based on the minimum makespan for lot production attained for the subset $\{1, \ldots, j-1\}$ of orders having a total size t_k on machine k.
The boundary conditions are as follows:

1. $f(0, 0, \ldots, 0) = 0 \qquad , j = 0$
2. $f(j, 0, \ldots, 0) = \left\lceil \frac{\sum_{i=1}^{j} o_i}{L} \right\rceil P \qquad , 0 < j \leq n$
3. $f(0, t_1, \ldots, t_k, \ldots, t_{m-1}) = \infty \qquad , \forall k : 1 \leq k \leq m-1, \quad 0 < t_k \leq S_n$

Condition 2 is the result of lot-scheduling all n orders on machine m. Condition 3 represents the empty set of orders. The recursion step is:
$f(j, t_1, \ldots, t_k, \ldots t_{m-1}) =$

$$\begin{cases} \max(f(j-1, t_1, \ldots, t_k, \ldots, t_{m-1}), \left\lceil \frac{O^m_{(j,t_1,\ldots,t_{m-1})}}{L} \right\rceil P) & \forall k \quad t_k < o_j \\ \min \left(\min\limits_{1 \leq k \leq m-1, t_k \geq o_j} \left(\max\limits_{1 \leq k \leq m-1} (f(j-1, t_1, \ldots, t_k - o_j, \ldots, t_{m-1}), \right. \right. & \text{otherwise} \\ \qquad \left. \left\lceil \frac{O^k_{(j,t_1,\ldots,t_k,\ldots t_{m-1})}}{L} \right\rceil P) \right), \\ \qquad \max(f(j-1, t_1, \ldots, t_{m-1}), \left\lceil \frac{O^m_{(j,t_1,\ldots,t_{m-1})}}{L} \right\rceil P) \Big) \end{cases} \tag{3}$$

1. When $t_k < o_j$ for all k, order j is scheduled on Machine m, M_m.
2. When there exists an index k so that its current size, t_k fulfills $t_k \geq o_j$, order j may be scheduled on Machine k, M_k. The makespan is computed by the maximum of all completion times on all m machines (the inner maximum condition). It is scheduled

on Machine k only if it achieves the minimum makespan of all machines, also examining the case that order j is scheduled on Machine m (the second component in the outer minimum function on the second condition).

The optimal solution is given by $min\Big(f(n, t_1, \ldots, t_{m-1})\Big)$, for every k, $1 \leq k \leq m-1$ and every $0 \leq t_k \leq S_n$.

Theorem 2 *The optimal makespan on $m \geq 2$ identical parallel machines can be solved in $\mathcal{O}(n \cdot {S_n}^{m-1})$ processing time.*

Proof The optimal makespan on $m \geq 2$ identical parallel machines is computed by the DP based on Equation 3. The recursive function is calculated for every job j, $0 \leq j \leq n$, and for every t, $0 \leq t \leq S_n$, for all machines, M_k, $0 < k \leq m-1$, except for the last machine, resulting in $\mathcal{O}(n \cdot {S_n}^{m-1})$ processing time. Reconstructing the solution is done by finding the minimum cost obtained in the final row, $f(n, t_1, \ldots, t_{m-1})$, which takes $\mathcal{O}({S_n}^{m-1})$ additional time, followed by backtracking to $f(0, 0, \ldots, 0)$, for an addition of $\mathcal{O}(n + mS_n)$, and a total of $\mathcal{O}(n \cdot {S_n}^{m-1})$ running time. ∎

4 Heuristic for Minimizing Makespan of m Identical Parallel Machines

The problem $Pm|lot, split|C_{\max}$ is of interest because minimizing the makespan has the effect of balancing the load over the various machines, which is an important objective in practice. In this section we will consider the heuristic described below:

1. Arrange the orders in any arbitrary sequence.
2. Iterate over all machines, locating Machine M_k with lowest sum of order sizes scheduled on it.
3. Place the current order on M_k obtained by Step 2.
4. If there are still unscheduled orders, go to Step 2.

The following theorem presents an upper bound on $\frac{C^H_{\max}}{C^{OPT}_{\max}}$, where $C^H_{\max}$ denotes the makespan of lot-scheduling according to the heuristic above, and $C^{OPT}_{\max}$ denotes the makespan of an optimal lot-schedule. This type of worst case analysis gives an indication of how well the heuristic is guaranteed to perform, as well as the type of instances for which the heuristic performs badly. Next, we prove a constant factor approximation of the heuristic to optimal.

Theorem 3 *For $Pm|lot, split|C_{\max}$:*

$$\frac{C^H_{\max}}{C^{OPT}_{\max}} \leq 2$$

Proof Let $\sum_{j \in M_k} o_j$ be the sum of sizes of all orders scheduled on machine k, $1 \leq k \leq m$. One can assume, without loss of generality, that the machines are arranged in non increasing order of their makespan, i.e. $\sum_{j \in M_1} o_j \geq \sum_{j \in M_2} o_j \geq \ldots \geq \sum_{j \in M_m} o_j$. Obviously,

$$\sum_{j \in M_1} o_j + \sum_{j \in M_2} o_j + \ldots + \sum_{j \in M_m} o_j = \sum_{j=1}^{n} o_j. \tag{4}$$

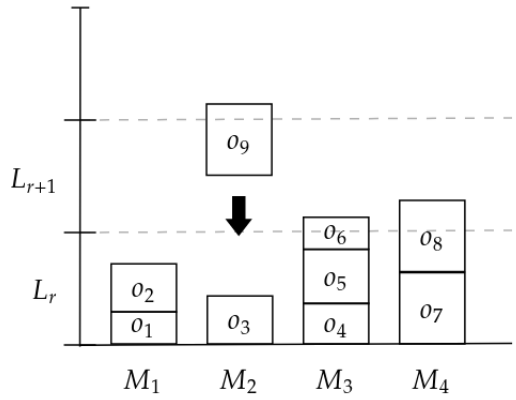


Fig. 2: According to the heuristic, order o_i is scheduled on machine M_2

Using the strategy of our heuristic , for every k, $k = 2, \ldots, m$, $\sum_{j \in M_1} o_j - \sum_{j \in M_k} o_j \leq o_{\max}$, then

$$\sum_{j \in M_1} o_j \leq \sum_{j \in M_k} o_j + o_{\max} \tag{5}$$

Combining equation 4 and 5 we receive that for $k = 2, \ldots, m$:

$$m \sum_{j \in M_1} o_j \leq \sum_{j=1}^{n} o_j + (m-1) o_{\max}$$

$$\sum_{j \in M_1} o_j \leq \frac{\sum_{j=1}^{n} o_j}{m} + \frac{m-1}{m} o_{\max} \tag{6}$$

By our initial arrangement, M_1 has the largest sum of order sizes, thus, $\sum_{j \in M_1} o_j = C^H_{\max}$, and

$$C^H_{\max} \leq \frac{\sum_{j=1}^{n} o_j}{m} + \frac{m-1}{m} o_{\max} \tag{7}$$

Obviously, the lower bound for $C^{OPT}_{\max}$ is given by $\frac{\sum_{j=1}^{n} o_j}{m}$. Using the assumption that $o_{\max} \leq L$, and that the processing of each lot is P, we obtain that $C^H_{max} \leq C^{OPT}_{max} + P$. Recall that I^{OPT} is used to denote the optimal number of lots for a given setting. The approximation factor is thus $\frac{C^H_{\max}}{C^{OPT}_{\max}} \leq \frac{C^{OPT}_{max}+P}{C^{OPT}_{\max}} = 1 + \frac{P}{I^{OPT} P} = 1 + \frac{1}{I^{OPT}}$. Using the fact that $1 + \frac{1}{I^{OPT}} \leq 2$, we conclude that the constant approximation factor is 2. ∎

To show that the approximation factor is tight, consider the following instance: $L = 10, P = 20$ and $o_j = \{2, 1, 2, 5, 10\}$, as illustrated in Figure 3. Applying the heuristic suggested above on two parallel identical machines will assign orders o_1 and o_4, with order sizes 2 and 5 to Machine M_1, and the remaining orders to Machine M_2 with order sizes 1, 2, and 10. A single lot is used on M_1, while two lots on M_2, for a total of $C_{\max} = 40$. However, an optimal solution may assign order o_5 with order size 10 to the first machine, leaving the remaining orders to the second machine, using only a single lot

on both machines. That is, $I^{OPT} = 1$, $C_{\max} = 20$, and $\frac{C^H_{\max}}{C^{OPT}_{\max}} = 1 + \frac{1}{I^{OPT}} = 1 + \frac{1}{1} = 2$, having a factor of 2 between both methods.

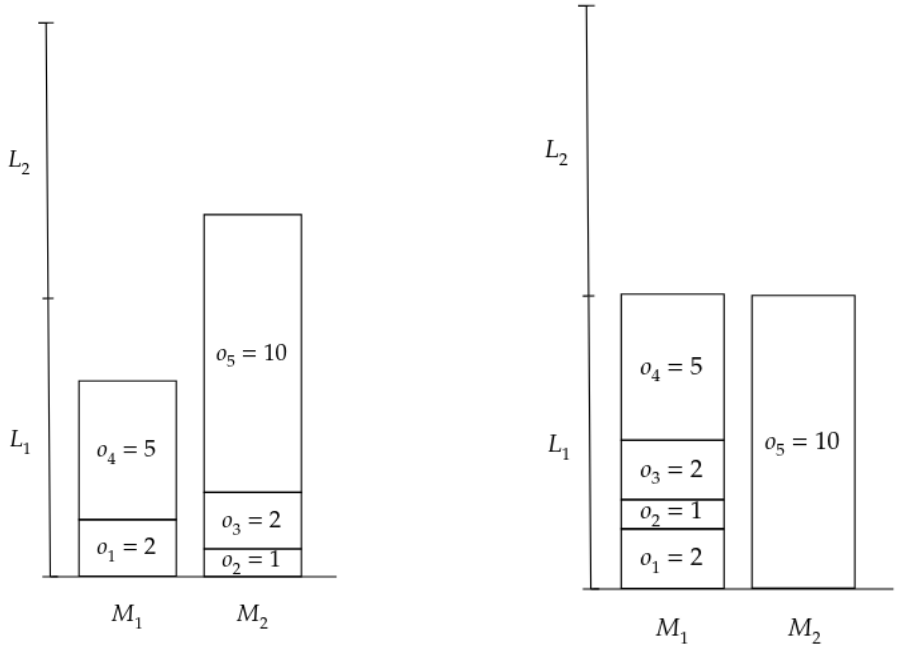


Fig. 3: Lower bound for the approximation factor of 2.

5 Conclusion

In this paper we study the problem of minimizing the makespan in case of lot scheduling, and propose a DP solution to $P_m|lot, split|C_{\max}$, and suggest a constant worst case performance approximation factor algorithm. This work concentrates on *serial splitting*, that is, when a given order is split into two consecutive lots. Additional splitting methods, other than serial splitting, which are more adequate for the current machine environment, will be considered in future work.

References

1. Jacek Blazewicz, Moshe Dror, and Jan Weglarz. Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, 51(3):283–300, 1991.
2. Cheng and Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292, 1990.
3. MR Garey and DS Johnson. Computer and intractability: A guide to the theory of np-completeness. freeman, new york, 1979.
4. Laleh Ghalami and Daniel Grosu. Scheduling parallel identical machines to minimize makespan: A parallel approximation algorithm. *Journal of Parallel and Distributed Computing*, 2018.

5. Laleh Ghalami and Daniel Grosu. Scheduling parallel identical machines to minimize makespan: A parallel approximation algorithm. *Journal of Parallel and Distributed Computing*, 133:221–231, 2019.
6. Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.
7. Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.
8. Yung-Tsung Hou, Dar-Li Yang, and Wen-Hung Kuo. Lot scheduling on a single machine. *Information Processing Letters*, 114(12):718–722, 2014.
9. Chung-Yee Lee, Reha Uzsoy, and Louis A Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775, 1992.
10. Yuanzhe Li, Laleh Ghalami, Loren Schwiebert, and Daniel Grosu. A gpu parallel approximation algorithm for scheduling parallel identical machines to minimize makespan. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 619–628. IEEE, 2018.
11. Daniel Oron, Dvir Shabtay, and George Steiner. Scheduling on identical parallel machines with controllable processing times to minimize the makespan. *extended*, page 61.
12. Onur Ozturk, Mehmet A Begen, and Gregory S Zaric. A branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize makespan. *International Journal of Production Research*, 55(6):1815–1831, 2017.
13. Michael L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Publishing Company, Incorporated, 5th edition, 2016.
14. Chris N. Potts. Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, 10(2):155–164, 1985.
15. Chris N. Potts and Mikhail Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249, 2000.
16. Ravi Sethi. On the complexity of mean flow time scheduling. *Mathematics of Operations Research*, 2(4):320–330, 1977.
17. Reha Uzsoy, Chung-Yee Lee, and Louis A Martin-Vega. A review of production planning and scheduling models in the semiconductor industry part i: system characteristics, performance evaluation and production planning. *IIE transactions*, 24(4):47–60, 1992.
18. Reha Uzsoy, Chung-Yee Lee, and Louis A Martin-Vega. A review of production planning and scheduling models in the semiconductor industry part ii: Shop-floor control. *IIE transactions*, 26(5):44–55, 1994.
19. Shijin Wang, Xiaodong Wang, Jianbo Yu, Shuan Ma, and Ming Liu. Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of cleaner production*, 193:424–440, 2018.
20. E Zhang, Ming Liu, Feifeng Zheng, and Yinfeng Xu. Single machine lot scheduling to minimize the total weighted (discounted) completion time. *Information Processing Letters*, 142:46–51, 2019.

MISTA 2019

Heuristics for the Min-Max Regret Shortest Path Problem under Interval Costs

Yang Lin · Wei Wu · Yannan Hu · Mutsunori Yagiura

Abstract The classical shortest path problem is a well-known problem with many applications. In the classical shortest path problem, every edge is assigned a cost, and the objective is to find a path from a start vertex to an end vertex with minimum cost. However, in real-world situations, costs of edges are usually uncertain. The min-max regret shortest path problem under interval costs (MMR-SP) is a generalization of the shortest path problem. We consider three heuristic algorithms for the MMR-SP. Our basic algorithm Fix-MMR finds first k shortest paths under a given scenario and returns the best path. The other two algorithms are based on the Fix-MMR. We propose an iterative method and a "kick" operation for updating the scenario in the Fix-MMR. The computational results show that the proposed algorithms are fast and work for general graphs. We then test our algorithms on random graphs, Karaşan's graphs (a directed acyclic graph), and other special graphs. We obtain better experimental results compared with the results from CPLEX and existing research.

Keywords Shortest path problem · Robust optimization · Min-max regret · Heuristic algorithm

1 Introduction

The classical shortest path problem is a well-known problem with many applications in the fields of navigation systems and others. In graph theory, the shortest path problem is defined as follows. For a given set of vertices V, set of edges E, start vertex $s \in V$, destination $t \in V$, and costs w_{ij} for each edge $(i, j) \in E$, the objective is to find a path between s and t such that the sum of the costs of the constituent edges is minimized.

Y. Lin · Y. Hu · M. Yagiura
Nagoya University, Japan
E-mail: linyang_77@yahoo.co.jp
E-mail: yannanhu@nagoya-u.jp
E-mail: yagiura@nagoya-u.jp

W. Wu
Seikei University, Japan
E-mail: wuwei@st.seikei.ac.jp

The classical shortest path problem can be solved in polynomial time by many algorithms such as Dijkstra's algorithm, Bellman-Ford algorithm, and Floyd-Warshall algorithm. Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge costs. The time complexity is $O(|V|^2)$, but it can be improved with a binary heap: the resulting time complexity will be $O((|E|+|V|)\log|V|)$. Bellman-Ford algorithm is suitable for single-source problems in which edge costs may be negative. The time complexity is $O(|V||E|)$. Additionally, the Shortest Path Faster Algorithm (SPFA) is an improvement of the Bellman-Ford algorithm. Although the worst-case complexity of the SPFA is the same as that of Bellman-Ford algorithm, the SPFA is believed to work well on random sparse graphs and is particularly suitable for graphs that contain negative-cost edges. Floyd-Warshall algorithm solves all-pairs shortest path problems with both positive and negative edge costs. The time complexity is $O(|V|^3)$.

However, in real-world situations, costs of the edges might be uncertain. Instead of fixed costs, we might only be given a range of costs. This study focused on a problem with uncertain edge costs that we call the min-max regret shortest path problem under interval costs (MMR-SP). In this problem, the objective function is more complicated than that of the shortest path problem, because we need a new way of estimating the value of a solution (a path). Note that we cannot simply sum up the costs, because we are only given the range of costs. A scenario in the MMR-SP is a cost vector where every edge is assigned a specific cost value from its given range. The regret of a path under a given scenario is the value difference between the cost of this path and the optimal value under the given scenario. Maximum regret of a path is the maximum regret for all the scenarios. The concept of regret is used for estimating the value of a path, which means that we prefer a less regretful path. We want to find a path such that even when the worst scenario happens, the cost between our path and the optimum path under the worst scenario is minimized.

The MMR-SP has been proved to be $\mathcal{NP}$-hard by Zielinski [18]. Although this problem can be formulated as a mixed integer programming (MIP) problem, it takes too much time to solve for large-scale instances.

For the MMR-SP, Kang proposed a heuristic algorithm designed for directed acyclic graphs in 2013 [7]. There are also exact algorithms designed by Montemanni et al. [12], and Montemanni and Gambardella [10,11]. Montemanni et al. [12] described an exact algorithm based on branch and bound framework. Montemanni and Gambardella [10] introduced an exact algorithm based on Benders decomposition. Karaşan et al. [8] proposed a method for removing useless edges in order to make algorithms faster. They also introduced an important directed acyclic graph called Karaşan's graph [8].

Montemanni and Gambardella [11] proposed an idea of finding k-shortest path for the case when every edge takes its maximum cost. The results were good for random graphs. Unfortunately, it reported poor performance on Karaşan's graphs.

However, even though some of the above exact algorithms are faster than an MIP solver, they take a long time to solve problems on Karaşan's graphs. Although the approach proposed in [7] is efficient and accurate, it only works on directed acyclic graphs.

In this paper, we consider three heuristic algorithms for solving the MMR-SP. We first examine a fixed-scenario method by solving the classical k-shortest path problem that aims to find the first k shortest paths. We then propose two iterative approaches based on k-shortest path algorithm. In each iteration, we consider finding scenarios such that the first k shortest paths are much closer to the optimal path in the MMR-SP.

The idea of finding such scenarios is based on heuristic methods. Our computational experiments show that the best path among first k shortest paths found under some fixed scenario such as median-scenario is almost as good as the optimum path for the MMR-SP. The results also show that in most cases, it is easier to obtain an optimal solution for the instances based on random graphs rather than Karaşan's graphs. We then conduct the experiments on Karaşan's graphs, and we obtain better experimental results compared with the results from CPLEX and existing research.

2 Problem description

Because the min-max regret shortest path problem under interval costs is a generalization of the classic shortest path problem, the existing algorithms for solving the shortest path problem can be used in some subroutines of our algorithms for the MMR-SP. In this chapter, we first introduce the classical shortest path problem.

2.1 The shortest path problem

Given a graph $G = (V, E)$, a start vertex $s \in V$, a terminal vertex $t \in V$, and non-negative cost of each edge $w_{ij} > 0$, $\forall (i, j) \in E$, where V is the set of vertices and E is the set of edges, the shortest path problem aims to find an s-t path such that the sum of the costs of its constituent edges is minimized.

Let $x_{ij} \in \{0, 1\}$ represent whether to choose edge (i, j) or not. The linear programming formulation of the shortest path problem [1] is

$$\min \quad \sum_{(i,j)\in E} w_{ij} x_{ij}, \tag{1}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} 1, & \text{if } i = s, \\ -1, & \text{if } i = t, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i \in V, \tag{2}$$

$$x_{ij} \geq 0 \qquad \forall (i, j) \in E. \tag{3}$$

The objective function (1) minimizes the total cost of the s–t path. The constraints (2) ensure that x forms a path from s to t. Term $\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji}$ represents the difference between out-degree and in-degree of vertex i. For start vertex s, there is only one edge out; thus, out-degree of s minus in-degree of s must be 1. Similarly, there is only one edge in for destination t; thus, out-degree of t minus in-degree of t must be -1. For the other edges, there is only one edge in and one edge out. Thus, out-degree minus in-degree must be 0 for those vertices.

Because of the total unimodularity of the constraint matrix, the integer optimal solution could be obtained by solving the resulting linear programming (LP) problem. However, compared to the general LP approach, the shortest path problem can be solved with more efficient algorithms, which we introduce in Section 3.

Next, we introduce the dual formulation of the shortest path problem. Let y_i be the dual variable for vertex i. The corresponding dual formulation of the shortest path

problem is

$$\max \ \ y_s - y_t, \tag{4}$$

$$\text{s.t.} \ \ y_i - y_j \leq w_{ij}, \qquad \forall (i,j) \in E. \tag{5}$$

Variable y_i can be viewed as the cost of the shortest path from s to i.

2.2 The min-max regret shortest path problem under interval costs (MMR-SP)

In the MMR-SP, all the costs of edges are given by intervals $\left[w_{ij}^-, w_{ij}^+\right]$, $\forall (i,j) \in E$, where $0 \leq w_{ij}^- \leq w_{ij}^+$. A scenario in the MMR-SP is a cost vector in which every edge is assigned a specific cost from its given range.

Definition 1 *Given an instance of the MMR-SP, a scenario δ defines a set of costs $w_{ij}^\delta \in \left[w_{ij}^-, w_{ij}^+\right]$, $\forall (i,j) \in E$ on graph G.*

Next, let us introduce the following notations:

- S is the set of all scenarios;
- X is the set of all paths from s to t, that is, $X = \{x \mid x \text{ satisfies } (2)\text{–}(3)\}$;
- $w_{ij}^\delta \in \left[w_{ij}^-, w_{ij}^+\right]$ is the cost of edge under scenario δ;
- $val(x,\delta) = \sum_{(i,j)\in E} w_{ij}^\delta x_{ij}$ is the cost of path x under scenario δ;
- $\min_{y\in X} val(y,\delta)$ is the cost of the shortest path from s to t under scenario δ.

Definition 2 *The regret of path x under scenario δ is*

$$r(x,\delta) = val(x,\delta) - \min_{y\in X} val(y,\delta). \tag{6}$$

The regret is measured for a given path under a given scenario. If we choose path x, and scenario δ occurs, then the regret is the difference between the cost of our path x and cost of the best path under scenario δ. Given a path x, the maximum regret of path x is

$$r_{\max}(x) = \max_{\delta\in S} r(x,\delta). \tag{7}$$

Value $r_{\max}(x)$ for path x is the maximum, or the worst, regret of path x under every possible scenario. Note that regret value $r(x,\delta)$ is non-negative from its definition.

The goal of the MMR-SP is to find a path such that the maximum regret of this path is minimized. A path with minimized maximum regret is meaningful for evaluating the path in the MMR-SP, because such a path has a cost close to the optimum path, even in its worst situation.

Definition 3 *The goal of the MMR-SP is to find a path such that*

$$\min_{x\in X} \max_{\delta\in S} \left\{ val(x,\delta) - \min_{y\in X} val(y,\delta) \right\} \tag{8}$$

is satisfied.

To simplify the objective function, we introduce the worst-case scenario [2,4] (whose roots are in Yaman et al. [17]).

Lemma 1 *The worst-case scenario of a path x is*

$$w_{ij}^{\theta(x)} = \begin{cases} w_{ij}^-, & \text{if } x_{ij} = 0 \\ w_{ij}^+, & \text{if } x_{ij} = 1 \end{cases} \tag{9}$$

$$= w_{ij}^- + (w_{ij}^+ - w_{ij}^-)x_{ij}, \tag{10}$$

where the worst-case scenario of a path x is denoted as $\theta(x)$.

The worst-case scenario is a scenario that assigns maximum cost w_{ij}^+ to every edge on path x and minimum cost w_{ij}^- to every edge that is not on path x.

By using the worst-case scenario lemma, the maximum regret of path x in (7) can be rewritten as

$$\begin{aligned} r_{\max}(x) &= val(x, \theta(x)) - \min_{y \in X} val(y, \theta(x)) \\ &= \sum_{(i,j) \in E} w_{ij}^+ x_{ij} - \min_{y \in X} val(y, \theta(x)). \end{aligned} \tag{11}$$

Therefore, to obtain maximum regret $r_{\max}(x)$ of path x, we can first construct the worst-case scenario $\theta(x)$, and then calculate the cost of path x under scenario $\theta(x)$: $val(x, \theta(x)) = \sum_{(i,j) \in E} w_{ij}^+ x_{ij}$, and the cost of the shortest path from s to t for scenario $\theta(x)$: $\min_{y \in X} val(y, \theta(x))$.

The time complexity of the former part is $O(|E|)$, but for the latter part it depends on the shortest path algorithm. In this study, we use Dijkstra's algorithm with a binary heap, and the time complexity for computing the maximum regret of a given path is $O((|E| + |V|) \log |V|)$.

By using the result in (11), Definition 3 can be written as

$$\min_{x \in X} \left\{ \sum_{(i,j) \in E} w_{ij}^+ x_{ij} - \min_{y \in X} val(y, \theta(x)) \right\}. \tag{12}$$

2.2.1 Mixed integer programming formulation of the MMR-SP

By using variable x and dual variable y in the classical shortest path problem, the mixed integer programming (MIP) formulation of the MMR-SP [12] is

$$\min \quad \sum_{(i,j) \in E} w_{ij}^+ x_{ij} - y_t \tag{13}$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in E} x_{ij} - \sum_{j:(j,i) \in E} x_{ji} = \begin{cases} 1, & \text{if } i = s, \\ -1, & \text{if } i = t, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i \in V, \tag{14}$$

$$y_s = 0 \tag{15}$$

$$y_j \le y_i + w_{ij}^- + (w_{ij}^+ - w_{ij}^-)x_{ij}, \qquad \forall (i,j) \in E, \tag{16}$$

$$y_j \le x_{ij} \in \{0, 1\}, \qquad \forall (i,j) \in E. \tag{17}$$

Constraints (14) are the same as constraints (2) in the shortest path problem, which ensure that x is a path from s to t. Note that we add binary constraints for variable x, because total unimodularity is broken by constraints (16).

Constraints (16) ensure that y_t is the cost of the shortest path from s to t under the worst-case scenario of path x when the optimum path is found. They also describe the relationship of distance between every two adjacent vertices.

The MMR-SP has been proved to be $\mathcal{NP}$-hard by Zielinski [18].

3 Algorithms for the shortest path and k shortest path problems

We first introduce the algorithms for solving the shortest path and k-shortest path problems. There are many efficient algorithms for solving the shortest path problems:

- Dijkstra's algorithm [5] solves the single-source shortest path problem for non-negative edge costs.
- Bellman-Ford algorithm solves the single-source problem if edge costs may be negative.
- A* search algorithm [6] solves the single-pair shortest path problem using heuristics to speed up the search.
- Floyd-Warshall algorithm solves all pairs of shortest paths.

The time complexity of the naive implementation of this algorithm is $O(|V|^2)$. A more efficient implementation by using a binary heap can reduce time complexity to $O((|V| + |E|) \log |V|)$. Although Dijkstra's algorithm with a Fibonacci heap has a better time complexity $O(|E| + |V| \log |V|))$, the algorithm using a binary heap performs well in most practical cases. In this work, we consider the shortest path problem with positive costs, therefore we use Dijkstra's algorithm to solve involving classical shortest path problem. We apply Dijkstra's algorithm with a binary heap to solve the classical shortest path problems.

The Shortest Path Faster Algorithm (SPFA) [13] is an improvement of the Bellman-Ford algorithm. Although both algorithms have the same worst-case complexity, the SPFA is believed to work well on random sparse graphs and is particularly suitable for graphs that contain negative-cost edges.

The A* search algorithm is often used for designing efficient search algorithms in games. In this work, we use the A* search algorithm for solving the k-shortest path problem.

3.1 Dijkstra's algorithm with a binary heap

We show the outline of Dijkstra's algorithm with the binary heap for the shortest path problem in Algorithm 1. Array dis records the cost of the shortest path from s to every vertex. Array $prev$ is used to record the previous edge corresponding to array dis. We can retrieve every edge on the shortest path according to array $prev$ by referring to $prev[t]$, $prev[prev[t]]$, and so on.

In Algorithm 1, the "while" loop will execute at most $|V|$ rounds, because there are at most $|V|$ vertices in the binary heap. The "for" loop will execute $|E|$ rounds in total, because every edge will be computed at most one time. The time complexity for updating one node in the binary heap is $O(\log |V|)$. In general, the time complexity is $O(|E| \log |V|)$ for the "for" loop and $O(|V| \log |V|)$ for the "while" loop. Therefore, the overall time complexity is $O((|E| + |V|) \log |V|)$.

Algorithm 1 Dijkstra's algorithm with a binary heap

```
dis[i] ← inf, ∀i ∈ V {Initialize distance from s to i}
prev[i] ← null, ∀i ∈ V {Initialize previous vertex of vertex i}
dis[s] ← 0
Initialize binary heap {Maintaining dis and top element is minimum}
while true do
    if heap is empty then
        break
    end if
    p ← heap.top() {Return vertex that has been used with minimum distance}
    heap.pop() {The time complexity is O(log |V|)}
    if p is inf then
        break {If p is unreachable, break}
    end if
    for all q, (p, q) ∈ E do
        dis[q] ← min {dis[q], dis[p] + w_pq}
        update heap if dis[q] is changed {time complexity is O(log |V|)}
        prev[q] = p if dis[q] is changed
    end for
    if p is equal to t then
        break {if p is destination, break}
    end if
end while
return dis and prev
```

A binary heap is a heap data structure that takes the form of a binary tree. The heap data structure maintains a set of keys. It supports operations of finding minimum key, deleting minimum key, inserting key, decreasing key, and so on. In Algorithm 1, we only use finding, deleting, and decreasing operation. Because every time when we update the distance, it can only be decreased, it is a decreasing operation. It takes $O(1)$ for the finding operation and $O(\log |V|)$ for other 2 operations with the binary heap.

Fibonacci heap has a better time complexity on the decreasing operation, which is $O(1)$ in amortized time. For Algorithm 1, 2-3 heap [15] is a good substitution for Fibonacci heap, because 2-3 heap has the same time complexity as Fibonacci heap on decreasing operation. If we use 2-3 heap, time complexity should be $O(|E|+|V|\log|V|)$, because updating the heap in the "for" loop will only take $O(1)$.

However, in practical use, we did not obtain a better performance with 2-3 heap comparing with the binary heap. The reason might be a lack of optimization of the code. Thus, we decided to use the binary heap rather than 2-3 heap.

3.2 k-Shortest Path Algorithm

The k-shortest path algorithm is an extension of the shortest path algorithm. The algorithm not only finds the shortest path, but also $k-1$ other paths in non-decreasing order of cost. The problem can be restricted to have the k shortest path without cycles (non-cycle k shortest path) or with cycles. In the MMR-SP, if an s-t path contains cycles, then it can be dominated by a path without those cycles. We show it in Lemma 2. This fact implies that we can focus on searching non-cycle paths without loss of optimality.

Lemma 2 *In the MMR-SP, if an s-t path x' contains cycles, then a path $\bar{x}$ without those cycles will have maximum regret no more than that of path x'.*

Proof Suppose path x' contains cycles and path $\bar{x}$ is the path based on x' without cycles. Let D be the set of edges on the cycles. Considering

$$\begin{aligned} & r_{\max}(x') - r_{\max}(\bar{x}) \\ = & \sum_{(i,j)\in E} w_{ij}^{+} x'_{ij} - \min_{x\in X} val(x,\theta(x')) - \left(\sum_{(i,j)\in E} w_{ij}^{+} \bar{x}_{ij} - \min_{x\in X} val(x,\theta(\bar{x})) \right). \end{aligned} \tag{18}$$

By the definition of D, we have

$$\sum_{(i,j)\in E} w_{ij}^{+} x'_{ij} - \sum_{(i,j)\in E} w_{ij}^{+} \bar{x}_{ij} = \sum_{(i,j)\in D} w_{ij}^{+}. \tag{19}$$

For the shortest path under the worst-case scenario of path x' and $\bar{x}$, we have

$$w_{ij}^{\theta(x')} \geq w_{ij}^{\theta(\bar{x})} \qquad \forall (i,j) \in E;$$

thus, the cost of the shortest path on $\theta(x')$ is greater than or equal to the cost of the shortest path on $\theta(\bar{x})$. For the same reason, the difference is no more than $\sum_{(i,j)\in D} w_{ij}^{+}$, i.e.,

$$0 \leq \min_{x\in X} val(x,\theta(x')) - \min_{x\in X} val(x,\theta(\bar{x})) \leq \sum_{(i,j)\in D} w_{ij}^{+}.$$

Then formula (18) can be written as

$$\begin{aligned} & \sum_{(i,j)\in D} w_{ij}^{+} - \min_{x\in X} val(x,\theta(x')) + \min_{x\in X} val(x,\theta(\bar{x})) \\ \geq & \sum_{(i,j)\in D} w_{ij}^{+} - \sum_{(i,j)\in D} w_{ij}^{+} \\ = & 0 \end{aligned}$$

Thus, $r_{\max}(x') \geq r_{\max}(\bar{x})$. Q.E.D.

Nowadays, the best algorithm for solving the k-shortest path problem has a time complexity of $O(|E| + |V|\log|V| + k)$ [3]. In this study, we use an A* algorithm in Algorithm 2 to find the first k shortest paths.

In Algorithm 2, Q is a priority queue. Each element in Q contains the current path, last vertex p of current path, cost of current path, $rev[p]$, where $rev[i]$ is the cost of shortest path from i to t. To compute rev, we need to compute the shortest path that starts from vertex t in the reversed graph of G. Queue Q arranges elements according to the *cost of current path*$+rev[p]$, where p is the current vertex. The time complexity of inserting element to Q is $O(\log|Q|)$. This can be done by a binary heap. Notice that we do not check on duplicate vertices in the path. Thus, Algorithm 2 might return a path with cycles. Although we can add a recorder to remember the past vertices for solving this problem, we must somehow optimize the space complexity and time complexity for remembering the paths or past vertices.

In our program, we do not store the whole path on every node in Q, because we need to store at most $|V|$ vertices. Instead, we store the previous node, thus it takes only $O(1)$ space for every node and takes $O(1)$ for adding nodes. We need to check for duplicate vertices if we want to remove the cycles. However, we have not found an efficient way of removing the cycles, because it is unworthy to check every vertex every time, and it takes $O(|V|)$ in the worst case. We cannot spend the same space constructing a data structure for checking vertices for every path either.

However, if we store the previous node, then we cannot remove any node in Q during the whole computation of k-shortest path, because old nodes might have important path information. Therefore, we need huge space if $|V|$ and k are large. Fortunately, a small number of k is enough for our algorithms. Cost of memory is acceptable.

Property 1 *In Algorithm 2, we can skip path x with last vertex p if we have examined more than k times pop out of p because path x cannot be strictly better than any of the first k shortest paths.*

Proof By contradiction, suppose path x has a lower cost than k-th shortest path, and there exists a vertex p such that, during the computation, s-p path (path from s to p) in x is examined after at least k other s-p paths in Q. Then we can construct path y by connecting other s-p path and p-t path in x. y costs less than or equal to x, because s–p part in path y is less than or equal to that in x. Similarly, we can construct at least k paths which cost less than or equal to path x. This truth is contradicted with the assumption that x has a lower cost than k-th shortest path.

Thus, we can skip this kind of path. Q.E.D.

Let κ be the maximum number of iterations for finding the shortest path in Algorithm 2. Then, the time complexity is $O(\kappa \log \kappa)$, because the insertion takes $O(\log |Q|)$ for Q. The space complexity is $O(\kappa)$.

With property 1, we are able to calculate κ. Because there are at most $|V|$ different vertices in a graph and every vertex will be popped out at most k times, this operation will be performed at most $k|V|$ times. As for insertion, popping out every vertex exactly once means $|E|$ times of insertion, thus there are at most $k|E|$ times of insertion. In total, $\kappa \leq k|E|$, because the number of insertions is always greater than the number of popping out operations.

The worst-case time complexity of Algorithm 2 is $O(k|E| \log k|E|)$, and space complexity is $O(k|E|)$.

4 Heuristics for the MMR-SP

In this section, we propose three heuristic algorithms for solving the MMR-SP: Fix-MMR, Iterated Fix-MMR (IFix-MMR), and Improved Iterated Fix-MMR (IIFix-MMR).

4.1 Fix-MMR

We show a heuristic approach, which we call Fix-MMR, based on the observation that, feasible solutions to an MMR-SP instance can be obtained by fixing a scenario, solving the corresponding SP instance, and evaluating the maximum regret of the obtained

Algorithm 2 k-shortest path algorithm with A*

```
Let Q ← ∅ be a priority queue {Element in Q consists of the current path, cost of current path, last vertex p of current path, rev[p]}
Reverse graph G and compute cost of the shortest path rev[i] from t to i, ∀i ∈ V
Q.add({s}, 0, rev[s]) {Add start vertex as the initial node to Q}
tot = 0 {Number of paths found}
ans ← ∅
count[i] ← 0, ∀i ∈ V {Record number of times that vertex i is popped out from Q}
while Q is not empty and tot < k do
    h ← Q.top()
    Q.pop()
    p ← last vertex in h.path
    count[p] ← count[p] + 1
    if count[p] > k then
        continue
    end if
    for ∀i ∈ V, (p, i) ∈ E do
        next ← ({h.path, i}, h.cur + w_pi, rev[i])
        Q.add(next)
        if i is equal to t then
            tot ← tot + 1
            ans ← ans ∪ {next}
        end if
    end for
end while
return ans
```

solutions. Similar approaches were used for other interval min-max regret problems (see, e.g., [9], [14], and [16]).

We show the steps of Fix-MMR(k,δ) as follows:

- Finding the first k shortest paths under scenario δ.
- Computing the maximum regret of these k paths.
- Returning the path with minimum-maximum regret among k paths.

The time complexity is $O(k(|E| + |V|) \log |V| + k|E| \log k|E|)$. We can see that, in the worst case, finding k shortest paths and computing the maximum regret of k paths require almost the same time.

To find a better path, we can either increase k or choose a better scenario δ. Because the number of paths in a graph might be exponential, increasing k is not an efficient way of finding an optimum path. We mainly focus on choosing a good scenario δ.

4.2 Iterated Fix-MMR (IFix-MMR)

The IFix-MMR we proposed iteratively calls Fix-MMR(k, δ) as a subroutine, and updates scenario δ based on the obtained solutions for next iteration.

First, we list the useful scenarios which we considered as an initial scenario for the IFix-MMR. For example, Montemanni and Gambardella [11] use the worst scenario by definition 4 for solving the MMR-SP.

Definition 4 *Worst scenario* δ^+*:* $w_{ij}^{\delta^+} = w_{ij}^+$.

Similarly, we can define the best scenario:

Definition 5 *Best scenario* δ^-*:* $w_{ij}^{\delta^-} = w_{ij}^-$.

We can also use median scenario by definition 6.

Definition 6 *median scenario* $\bar{\delta}$*:* $w_{ij}^{\bar{\delta}} = \frac{1}{2}(w_{ij}^- + w_{ij}^+)$.

By median scenario lemma in survey [4], we have:

Lemma 3 *Given an instance* I *in the MMR-SP. Let* $\bar{x}$ *be the shortest path under median scenario* $\bar{\delta}$*, and* x^* *be the optimum path of instance* I*. Then* $r_{\max}(\bar{x}) \leq 2r_{\max}(x^*)$.

Lemma 3 implies that if we try Fix-MMR(1,$\bar{\delta}$), the path we find will have an approximate rate of 2.

Furthermore, we consider scenario δ_α as follows:

Definition 7 *Scenario* δ_α $(0 \leq \alpha \leq 1)$ *is a scenario such that*

$$w_{ij}^{\delta_\alpha} = \alpha w_{ij}^- + (1-\alpha) w_{ij}^+. \tag{20}$$

Note that the worst scenario δ^+, the best scenario δ^-, and the median scenario $\bar{\delta}$ can be considered as special cases of δ_α where $\alpha = 0, 1$ and 0.5, respectively. A warm start scenario δ_{α^*} for the Fix-MMR is determined by Algorithm 3. In Algorithm 3, we use α_l and α_u as the lower bound and upper bound of α. The algorithm aims to obtain best scenario α^* in $\{\alpha = \alpha_l + i\Delta \mid \alpha \leq \alpha_u \text{ and } i \in \mathbb{Z}_{\geq 0}\}$. Algorithm 3 costs time $O\left(\frac{(\alpha_u - \alpha_l)}{\Delta}(|V| + |E|)\log|V|\right)$. In this study, we set the algorithm parameters α_l, α_u, and Δ as follows: $\alpha_l = 0, \alpha_u = 1$ and $\Delta = 0.1$.

Algorithm 3 Deciding α^*

1: Given value α_l, α_u and Δ.
2: **for** $\alpha = \alpha_l; \alpha \leq \alpha_u; \alpha = \alpha + \Delta$ **do**
3: Construct scenario δ_α by Definition 7.
4: Find the shortest path x_α under scenario δ_α.
5: Compute maximum regret $r_{\max}(x_\alpha)$ of path x_α.
6: **if** $r_{\max}(x_\alpha) < r_{\max}(x_{\alpha^*})$ **then**
7: $\alpha^* \leftarrow \alpha$.
8: **end if**
9: **end for**
10: **return** α^* with minimum $r_{\max}(x_\alpha^*)$.

For updating the current scenario, we define scenario generation function $f(x, \delta)$ such that

$$w_{ij}^{f(x,\delta)} = \begin{cases} w_{ij}^{\delta}, & \text{if } x_{ij} = 0, \\ p_0 \cdot w_{ij}^{\delta}, & \text{otherwise}, \end{cases} \tag{21}$$

where x is a path, δ is a scenario, and p_0 is a random variable.

In this study, we recommend a uniform random variable p_0 from 0.5 to 1. Function f means constructing a new scenario from δ such that costs of edges that are on path x become $p_0 w^\delta$, and costs of the other edges remain the same. We store the current best path as $x^{\text{best}} = \arg\min_i r_{\max}(x^i)$. Note that the updating scenario obtained by function f may be less than w_{ij}^- for some edge (i, j). In this study, we also adopt such scenarios.

We show the overall IFix-MMR approach in Algorithm 4. Time complexity of IFix-MMR depends on the number of times it calls Fix-MMR. We lower the cost on the current best path in a new scenario in order to make the first k shortest paths return a better maximum regret. A random variable p_0 is better than a fixed coefficient and allows for more different scenarios.

Algorithm 4 IFix-MMR

Generate an initial scenario δ and obtain solution x by Algorithm 3.
$x^{\text{best}} \leftarrow x$.
repeat
  $\delta \leftarrow f(x, \delta)$.
  Solve Fix-MMR(k, δ) and obtain solution x.
  if $r_{\max}(x) < r_{\max}(x^{\text{best}})$ **then**
    $x^{\text{best}} \leftarrow x$.
  end if
until time limit is reached
return x^{best} with minimum $r_{\max}(x^{\text{best}})$.

4.3 Improved iterated Fix-MMR (IIFix-MMR)

We further consider an improved approach IIFix-MMR in which a penalty scenario is considered when there is no improvement after the previous iteration. At first, IIFix-MMR works the same as IFix-MMR. Once it cannot improve the current solution, it generates a new penalty scenario δ^{penalty}. Then, the updating function f will update the scenario based on new δ^{penalty}. Given scenario δ and U, the set of edges in k shortest paths in the previous iteration, we define penalty scenario generation function$g(\delta, U)$ as follows:

$$w_{ij}^{g(\delta,U)} = \begin{cases} p_1 \cdot w_{ij}^{\delta^0}, & \text{if } (i,j) \in U, \\ w_{ij}^{\delta^0}, & \text{otherwise,} \end{cases}$$

where p_1 is a random variable. In this study, we recommend a uniform random variable p_1 from 1 to 2.

The main idea of this modification is to construct a new scenario such that costs of the visited edges are increased. Because there is no improvement after the last iteration, we want to find paths that are different from previous k paths. We show the IIFix-MMR approach in Algorithm 5.

4.4 Tabu list

We noticed that some paths from s to t could be obtained multiple times in IFix-MMR and IIFix-MMR, because we apply Fix-MMR multiple times In such a case, we do not need to compute the maximum regret of the same path. Therefore, we consider a hash table to store the paths we have checked and skip the maximum regret computation if a path is inside the hash table.

Algorithm 5 IIFix-MMR

Generate an initial scenario δ^{init} and obtain solution x by Algorithm 3.
$x^{\text{best}} \leftarrow x$ and $\delta^{\text{penalty}} \leftarrow \delta^{\text{init}}$.
repeat
  $\delta \leftarrow f(x, \delta^{\text{penalty}})$.
  Solve Fix-MMR(k, δ), obtain the best solution x, and reset U as the set of edges in k shortest path.
  if $r_{\max}(x) < r_{\max}(x^{\text{best}})$ **then**
    $x^{\text{best}} \leftarrow x$.
  else
    $\delta^{\text{penalty}} \leftarrow g(\delta^{\text{init}}, U)$.
  end if
until time limit is reached
return x^{best} with minimum $r_{\max}(x^{\text{best}})$.

Definition 8 *The hash code of path x is*

$$\Pi_{x_{i,j}=1 \wedge (i,j) \in E} I_{(i,j)} \mod P, \tag{22}$$

where P is a large prime, $I_{(i,j)} \geq 1$ is an integral identity number of edge (i, j), usually it is from 1 to $|E|$.

According to this definition, if we add one edge to the path, we do not need to recompute the hash code from the beginning. This means $O(1)$ for updating the hash code when adding a new edge.

We use a hash set to maintain the hash code. Every time a path from s to t is found, we check its presence in the hash set and update the hash set. It will only cost additional $O(k \log \kappa)$ when we call Fix-MMR. Value κ is the maximum capacity of the hash set.

5 Computational experiments

5.1 Instance generation

A Karaşan's graph [8] is composed of layers. Each layer has the same number of vertices, and all the vertices in the same layer is not connected, while the edges are fully connected from one layer to the next. The width of the graph is the number of vertices in each layer. An example of a Karaşan's graph with layer 3 and width 3 is shown in Figure 1.

Rules for generating the costs with lower bound l_a and upper bound l_b are defined as follows:

- $(1 - d)c \leq l_a \leq (1 + d)c$,
- $l_a + 1 \leq l_b \leq (1 + d)c$,
- $d \in [0.3, 0.9]$,
- $c \in [10, 20]$,

where d and c are parameters of the graph.

Some interesting properties are observed when generating graphs with different parameters [7]. For example, a graph with parameters $c = 20$, $d = 0.9$ and width=4 is more difficult than a graph with parameters $c = 20$, $d = 0.3$ and width=4. Here,

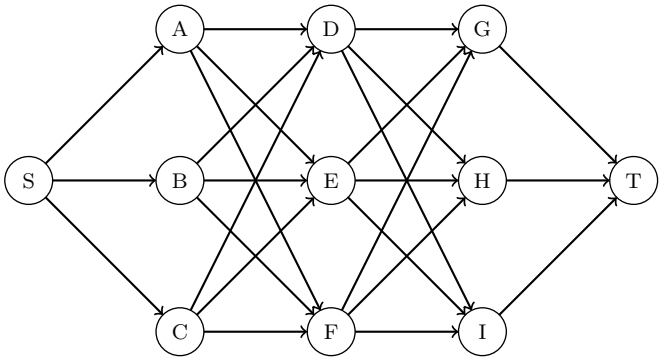


Fig. 1 Karaşan's graph with layer 3, width 3

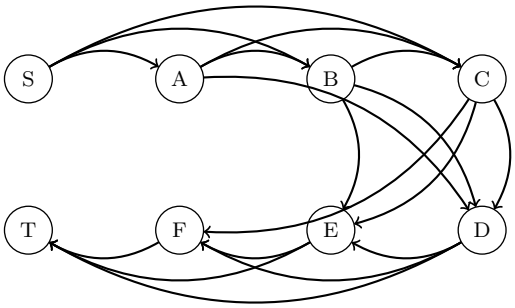


Fig. 2 Other graph with 8 vertices, and width 3

"difficult" means that mixed integer programming solver CPLEX requires more time to solve this problem on average.

A random general graph will be generated according to the maximum cost of edge $c_{\max}$ and the probability of generating an edge. Cost of every edge will be generated in the interval $[0, c_{\max}]$.

We also use other kinds of directed acyclic graphs, which are slightly different from Karaşan's graph. Parameters are the same as for Karaşan's graph, but there is no layer, and every vertex connects to certain next vertices. For example, in Figure 2 with width 3, every vertex connects to the next 3 vertices. We call this kind of graphs K* graphs.

5.2 Experimental environment

The algorithm parameters are defined as follows:

- $k = 500$, finding the first k shortest paths;
- $\alpha_l = 0.2$ and $\alpha_u = 0.8$, range of α for deciding scenario δ_α;
- $\Delta = 0.1$, which is the interval when enumerating the α;
- p_0 is a uniform random number from 0.5 to 1;
- p_1 is a uniform random number from 1 to 2;
- maximum number of elements in tabu list is 10^6;

– maximum capacity of priority queue Q is 2×10^7.

The algorithms are coded in C++ using the Standard Template Library. All the tests were done on a personal computer with Intel(R) Core(TM) i7-4980HQ CPU @ 2.80 GHz/16 GB main memory. We use ILOG CPLEX 12.8.0.0 to obtain an optimal solution of the MMR-SP by solving MIP showed in formulation (13)–(17).

The run time of algorithms Fix-MMR, IFix-MMR, and IIFix-MMR is calculated using the timestamp of the last update. Because algorithm IFix-MMR and IIFix-MMR would not stop running until they exceed the time limit, we use the timestamp of the last update to estimate the actual computing time. This also helps us understand the convergence speed of three algorithms.

In our C++ program, we start Fix-MMR, IFix-MMR, and IIF-MMR with $srand(0)$ to ensure that the random number sequence is the same every time we run the program.

5.3 Experimental results

5.3.1 Results on Karaşan's graphs

In research [7], a heuristic algorithm is proposed for a directed acyclic graph and tested in Karaşan's graph. Table 1 shows comparison results. Tests were done with 50 random samples for the types of graphs. The type parameter were noted in column "Instance." For example, notation "K-100-4-10-0.3" in column "Instance" means that this type of Karaşan's graph instances has layer 100, width 4, $c = 10$ and $d = 0.3$. Columns "Time" show the average run times of our algorithms, the heuristic algorithm in [7] and CPLEX; Columns "Gap" and columns "Ratio" show the average gap and the correct ratio of our algorithms and the heuristic algorithm in [7]. The correct ratio is calculated by

$$\text{Correct ratio} = \frac{\text{Number of cases that algorithm returns an optimum solution}}{\text{Number of test cases}}.$$

The optimum solutions are obtained by CPLEX. The algorithm showed in [7] is tested under the following environment: Intel Core2Duo @ 2.33 GHz CPU/1 GB main memory.

The results show that our algorithms have a better average gap and correct ratio compared with the algorithm in [7]. Both the IFix-MMR and the IIFix-MMR obtained optimal or near-optimal solutions for all the tested instances. For the IIFix-MMR, the results show that its correct ratio is more than 94%, and the average gap is below 0.02% for all the instances.

5.3.2 Results a general random graph

We also test algorithms on a general random graph and compare the results with [11]. The latter one is tested under Pentium 4 1.5GHz processor and 256MB of memory from [11]. Table 2 shows the results on random graph with 7000 vertices and maximum cost 100 for each edge. The probability of generating one pair of edge is 0.001. Tests were done with 50 random samples for this graph.

We found out that the MMR-SP is a lot easier for general graphs than Karaşan's graphs because even our worst algorithm Fix-MMR finds the optimum solution immediately during the experiments. The MIP solver CPLEX still requires a long time for

Table 1 Results for Karaşan's graphs

	CPLEX	[7]*			Fix-MMR		
Instance	Time	Time	Gap	Ratio	Time	Gap	Ratio
K-100-2-10-0.3	0.38	0.24	0.81%	61%	0.08	0.03%	96%
K-100-2-10-0.9	0.44	0.18	0.99%	28%	0.09	0.31%	58%
K-100-2-20-0.3	0.22	0.24	1.06%	52%	0.08	0.00%	100%
K-100-2-20-0.9	0.46	0.18	1.03%	29%	0.09	0.34%	62%
K-100-4-10-0.3	1.82	0.38	1.08%	24%	0.09	0.07%	90%
K-100-4-10-0.9	4.76	0.39	1.04%	10%	0.10	0.41%	36%
K-100-4-20-0.3	1.40	0.39	1.18%	28%	0.08	0.00%	98%
K-100-4-20-0.9	2.84	0.40	0.79%	21%	0.09	0.41%	46%
		IFix-MMR			IIFix-MMR		
Instance		Time	Gap	Ratio	Time	Gap	Ratio
K-100-2-10-0.3		0.08	0.00%	100%	0.08	0.00%	100%
K-100-2-10-0.9		0.18	0.00%	98%	0.17	0.00%	100%
K-100-2-20-0.3		0.08	0.00%	100%	0.08	0.00%	100%
K-100-2-20-0.9		0.18	0.00%	100%	0.18	0.00%	100%
K-100-4-10-0.3		0.10	0.01%	98%	0.10	0.01%	98%
K-100-4-10-0.9		0.28	0.02%	94%	0.26	0.02%	94%
K-100-4-20-0.3		0.08	0.00%	100%	0.08	0.00%	100%
K-100-4-20-0.9		0.25	0.11%	88%	0.25	0.00%	96%

Table 2 Results on a random graph with 7000 vertices and maximum cost 100 for each edge

	[11]*	Fix-MMR	IFix-MMR	IIFix-MMR	CPLEX
Time(s)	104.32	0.12	0.10	0.10	729.40
Average Gap	2.04%	0.00%	0.00%	0.00%	-
Correct Ratio	84%	100%	100%	100%	-

solving it to optimality. It spends 90% of time doing the pre-process; thus, there is no way for us to retrieve the solution early. The result indicates that the IFix-MMR and the IIFix-MMR are more powerful in solving the instances on general random graphs.

5.3.3 Results on K graphs*

We tested algorithms K* graphs. Table 3 shows the run time and the optimality gap of algorithms. Instance "*aaa-b*" in column "Instance" means the the corresponding graph contains *aaa* vertices and the width is *b*. Parameter *c* and *d* are random. We generate 5 instances for every type of parameters. Tests were done with 50 random samples for each graph. Columns "Time" and "Gap" show the average run time in seconds and the average optimality gap for each instance, respectively. The time limit for CPLEX is 1 hour (because of the pre-processing, we cannot stop CPLEX at exactly 3600 s). For the IFix-MMR and IIFix-MMR, we set time limits 2, 6, 15 and 150 seconds to instances according vertex number 500, 1000, 2000 and 4000, respectively.

From the results, we can see that CPLEX cannot return optimal solutions for instances with 4000 vertices, and widths 4, 5, and 6 within 1 hour. This result shows that the IIFix-MMR always returns a good solution with gap no more than 0.3% compared to CPLEX. For graphs with nodes less than 500, The IIFix-MMR always returns the optimal or near-optimal solution. For the IFix-MMR, it returns a solution with gap no more than 0.6% compared to CPLEX. This means that the "kick" operation in algorithm IIFix-MMR makes good improvements.

Table 3 Average gap and average run time for tested algorithms

	Fix-MMR		IFix-MMR		IIFix-MMR		CPLEX	
Instance	Gap	Time	Gap	Time	Gap	Time	Gap	Time
500-2	1.20%	0.1	0.06%	2	0.00%	2	0.00%	1.4
500-3	0.72%	0.1	0.00%	2	0.00%	2	0.00%	2.6
500-4	0.20%	0.1	0.00%	2	0.00%	2	0.00%	2.8
500-8	0.00%	0.1	0.00%	2	0.00%	2	0.00%	4.4
1000-2	2.01%	0.3	0.20%	6	0.02%	6	0.00%	7.0
1000-3	1.46%	0.2	0.31%	6	0.03%	6	0.00%	9.4
1000-4	1.09%	0.2	0.02%	6	0.05%	6	0.00%	26.4
1000-8	0.07%	0.2	0.00%	6	0.00%	6	0.00%	26.2
2000-2	2.55%	0.5	0.42%	15	0.11%	15	0.00%	26.2
2000-3	1.96%	0.4	0.29%	15	0.09%	15	0.00%	97.8
2000-4	1.98%	0.4	0.42%	15	0.28%	15	0.00%	234.6
2000-8	1.00%	0.3	0.12%	15	0.13%	15	0.00%	550.8
4000-2	2.62%	1.2	0.59%	150	0.02%	150	0.00%	201.6
4000-3	2.66%	0.9	1.03%	150	0.62%	150	0.58%	3,056.0
4000-4	3.79%	0.8	2.16%	150	1.92%	150	1.80%	3,758.1
4000-5	3.58%	0.8	1.72%	150	1.40%	150	1.30%	3722.3
4000-8	3.33%	0.8	2.42%	150	2.28%	150	2.25%	3,840.8

Let us examine the details in Figures 3, 4, 5, 6, and 7. The x axis is the processing time, and y axis is the gap compared with the best results from CPLEX. The red line the is curve of CPLEX, while the green line represents IFix-MMR, and the blue line represents IIFix-MMR. We can see that IFix-MMR and IIFix-MMR start returning the first solutions immediately but CPLEX needs a long time for pre-processing before returning the first solution. Even though CPLEX converges faster than the other two algorithms after pre-processing, IIFix-MMR still completes execution before CPLEX.

The IFix-MMR almost stops converging in the middle of processing, but IIFix-MMR converges nearly to 0 compared to CPLEX. Noticed that for test case with 4000 vertices, width 2 and width 4, our time limit is 150 s in Table 3. But IIFix-MMR converges within 80 s in Figure 4 and 40 s in Figure 7. The "kick" operation in IIFix-MMR makes a great progress to IFix-MMR, according to the test results.

As the width of test data increases from 2 to 4 or 8, our algorithms converge faster compared to CPLEX in Figure 5.

5.4 Other results

5.4.1 Test of Fix-MMR on different scenarios

We tested three representative scenarios for the Fix-MMR: the best scenario, the median scenario, and the best scenario. In Table 4, we test three scenarios on 2 different kinds of graphs by computing only the maximum regret of the shortest path.

Table 4 Average gap of Fix-MMR(1,δ) under different scenario

	Scenario		
Graph	Worst Scenario	Best Scenario	median scenario
Random Graphs	240.65%	1.15%	3.76%
Karaşan's graph	97.66%	3.09%	4.83%

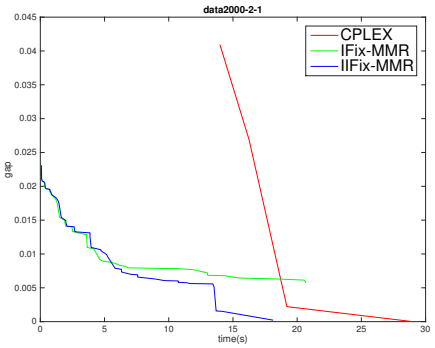


Fig. 3 Data 2000-2-1

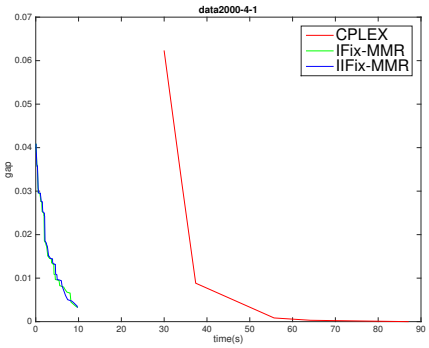


Fig. 6 Data 2000-4-1

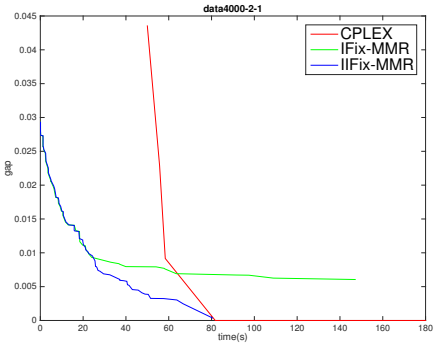


Fig. 4 Data 4000-2-1

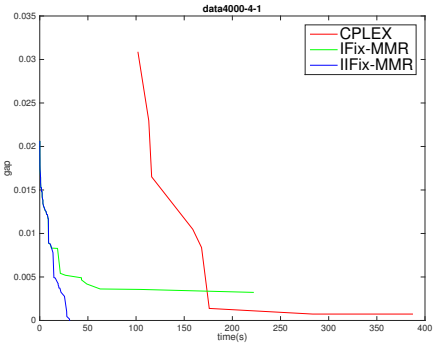


Fig. 7 Data 4000-4-1

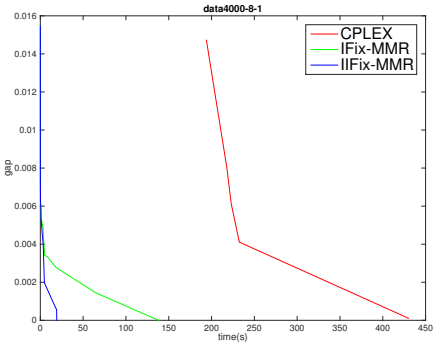


Fig. 5 Data 4000-8-1

According to the experiment, the median scenario is often better than the worst scenario in the Fix-MMR. The worst scenario is awful, and the best scenario is good for most cases.

Table 5 Average tabu list ratios

Instance	Time (s)	Duplicated Ratio	
		IFix-MMR	IIFix-MMR
K-100-4-20-0.9	1	52.43%	37.82%
K-1000-4-20-0.9	30	19.54%	8.42%
K-1000-4-20-0.9	60	28.38%	39.30%

5.4.2 Effect of using tabu list

We test the effect of our tabu list improvement on Karaşan's graph with 1000 layers and width 4 in Figure 8. The ratio Figure 8 is the percentage of duplicated paths. At first, most of paths are new, and no more than 5% of paths are duplicated before 40 s in the test data. As time elapses, duplicated ratio increases. This means that the chance for finding new paths in our algorithms is decreasing.

In Table 5, we test the average duplicated ratio on different size of graph with different time. Therefore, it is meaningful to apply tabu list improvement.

Fig. 8 The effect of using tabu list on instance K-1000-4

5.4.3 Test under different k

We test our three algorithms with different k on graph K-1000-4-20-0.9. Time limit for each algorithm is 60 seconds. Figure 9, 10 and 11 are our test results. The x axis is the time. The y axis is the min-max regret, and lesser values are better. We choose k as 20, 100, 500, and 2500 for each algorithm.

In Figure 9 with the Fix-MMR algorithm, we did not see significant improvements. This result explains why it is better to select good scenarios rather than increasing k for the Fix-MMR algorithm.

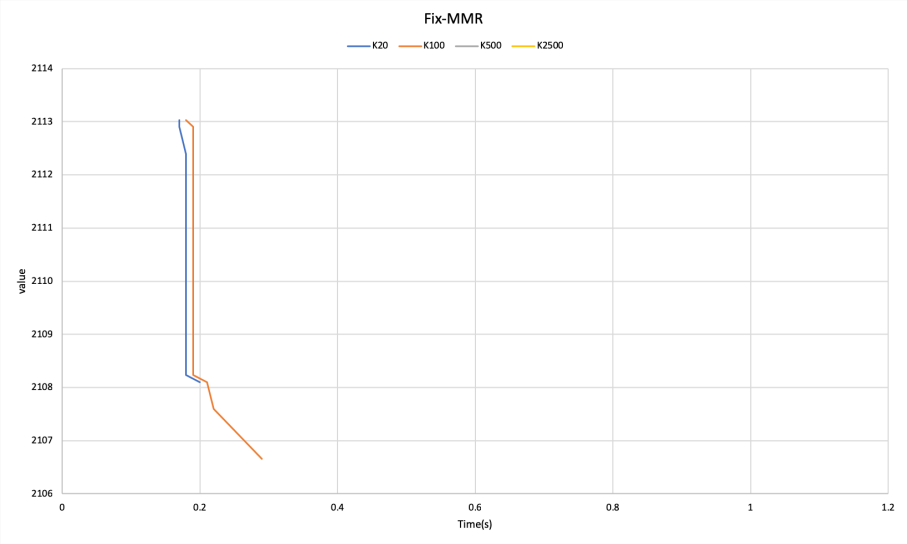


Fig. 9 K-curve for the Fix-MMR

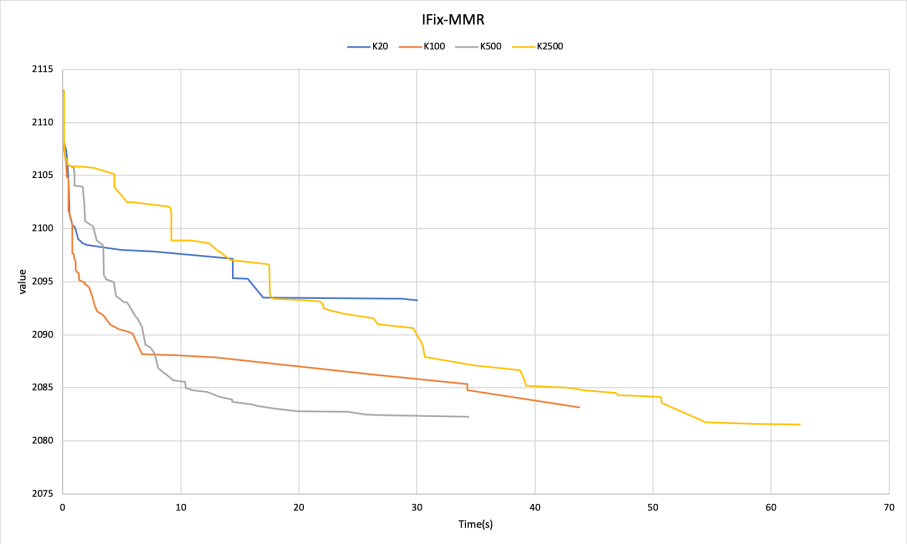


Fig. 10 K-curve for the IFix-MMR

For Figure 10, we observed that a larger k is more likely to help find a better solution if time is enough. However, a large k means low convergence speed, and we need more time for each iteration. For example, curve $k = 2500$ spends more time to achieve the same result compared to curve $k = 500$.

For Figure 11, it is hard to say which k will get a better solution because curves $k = 20$ and $k = 500$ return similar quality solutions. Maybe k is not an important

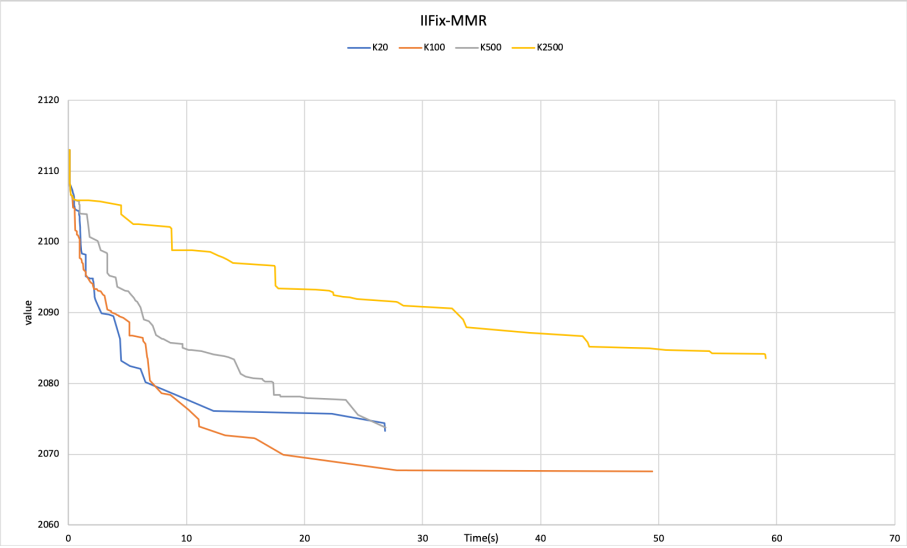


Fig. 11 K-curve for the IIFix-MMR

factor to the quality of the solution in IIFix-MMR algorithm. We might care more about time when we choose k.

5.4.4 Test under different parameter p_0 and p_1

In the IFix-MMR and the IIFix-MMR, p_0 and p_1 are uniform random variables. We tested the effect of setting ranges for p_0 and p_1. Table 6 shows the results of the IFix-MMR under different parameter p_0 on graph K-1000-4-20-0.9 within 30 seconds. Table 7 shows the test result of the IIFix-MMR under different parameters p_1 on graph K-1000-4-20-0.9 within 30 seconds. The average gap is computed by 50 executions according to the best lower bound provided by CPLEX within 1800 seconds.

Table 6 Results of the IFix-MMR (30 seconds) with different p_0 for Instance K-1000-4-20-0.9

	p_0					
	$[0, 0.5]$	$[0.5, 1]$	$[1, 1.5]$	$[0, 1]$	$[0.3, 0.7]$	$[0.5, 1.5]$
Average Gap	0.87%	0.54%	1.49%	0.42%	0.78%	1.49%

Table 7 Results of the IIFix-MMR (30 seconds) with different p_1 for Instance K-1000-4-20-0.9

	p_1			
	$[1, 2]$	$[1, 3]$	$[2, 3]$	$[1, 1.5]$
Average Gap	0.19%	0.19%	0.20%	0.16%

Although results on different kinds of graphs might be different, we can see that choosing $p_0 > 1$ is a really bad idea for the IFix-MMR. We found that parameter p_0 from 0.5 to 1 is the second-best choice in Table 6, while the best one is p_0 from 0 to 1.

For p_1 on the IIFix-MMR, there is not much difference between parameter p_1 from 1 to 2, 1 to 3, and 2 to 3. p_1 from 1 to 1.5 seems better compared to others on the IIFix-MMR.

6 Conclusion

The shortest path problem is known as a $\mathcal{P}$ problem. In this work, we studied the min-max regret shortest path problem under interval costs (MMR-SP), known as an $\mathcal{NP}$-hard problem. The MMR-SP aims to obtain a path to minimize the maximum regret, instead of a path with the minimum cost, as in the shortest path problem.

We proposed three heuristic algorithms to solve the MMR-SP: Fix-MMR, Iterated Fix-MMR (IFix-MMR), and Improved Iterated Fix-MMR (IIFix-MMR). The basic idea of the Fix-MMR(k,δ) is to find the first k shortest paths for a given scenario δ and return the path with min-max regret among these k paths, where k is an algorithm parameter. The time complexity is $O(k(|E| + |V|) \log |V| + k|E| \log k|E|)$. We also proposed a method to obtain a good scenario for the Fix-MMR. Although the median scenario is a good choice in most cases, the proposed method provides a more flexible choice for deciding on a scenario according to a given parameter α. We used an A* algorithm for solving the k-shortest path problem.

In the IFix-MMR, we update the scenario in each iteration after solving Fix-MMR to find a better solution, rather than simply increasing the number of shortest paths k. The experiment results showed that updating the scenario based on the obtained solutions is more efficient than increasing k in a Fix-MMR approach. The the IIFix-MMR algorithm embeds a "kick" operation to the IFix-MMR before updating the scenario when there are no improvements after the last iteration. The "kick" operation generates a penalty scenario based on the initial scenario and edges that have been passed on the previous iteration. This penalty scenario is assigned with larger costs on the passed edges.

The experiment results show that our proposed algorithms are better than the existing research [7] on Karaşan's graph. Compared to CPLEX, IIFix-MMR achieves a better convergence rate. The average optimality gap of IIFix-MMR is no more than 0.3%. On general graphs, our algorithms are better than those of existing research [11], according to the experimental results.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)
2. Aissi, H., Bazgan, C., Vanderpooten, D.: Minmax and minmax regret versions of combinatorial optimization problems: A survey. European Journal of Operational Research **197**, 427–438 (2009)
3. Aljazzar, H., Leue, S.: K*: A heuristic search algorithm for finding the k shortest paths. Artificial Intelligence **175**(18), 2129–2154 (2011)
4. Beyer, H.G., Sendhoff, B.: Robust optimization–a comprehensive survey. Computer methods in applied mechanics and engineering **196**(33-34), 3190–3218 (2007)
5. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1), 269–271 (1959)
6. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
7. Kang, J.G.: The minmax regret shortest path problem with interval arc lengths. International Journal of Control and Automation **6**(5), 171–180 (2013)
8. Karasan, O.E., Pinar, M.C., Yaman, H.: The robust shortest path problem with interval data. Tech. rep., Bilkent University, Ankara, Turkey (2001). Available from the web site of Optimization Online (http://www.optimization-online.org/DB_HTML/2001/08/361.html, accesed on June 1, 2019)

9. Kasperski, A., Zielinski, P.: An approximation algorithm for interval data minmax regret combinatorial optimization problems. Information Processing Letters **97**, 177–180 (2006)
10. Montemanni, R., Gambardella, L.: The robust shortest path problem with interval data via benders decomposition. 4OR **3**, 315–328 (2005)
11. Montemanni, R., Gambardella, L.M.: An exact algorithm for the robust shortest path problem with interval data. Computers & Operations Research **31**(10), 1667–1680 (2004)
12. Montemanni, R., Gambardella, L.M., Donati, A.V.: A branch and bound algorithm for the robust shortest path problem with interval data. Operations Research Letters **32**(3), 225–232 (2004)
13. Moore, E.F.: The shortest path through a maze. In: Proc. Int. Symp. Switching Theory, 1959, pp. 285–292 (1959)
14. Pereira, J., Averbakh, I.: The robust set covering problem with interval data. Annals of Operations Research **207**, 217–235 (2013)
15. Takaoka, T.: Theory of 2-3 heaps. In: International Computing and Combinatorics Conference, pp. 41–50. Springer (1999)
16. Wu, W., Iori, M., Martello, S., Yagiura, M.: Exact and heuristic algorithms for the interval min-max regret generalized assignment problem. Computers & Industrial Engineering **125**, 98–110 (2018)
17. Yaman, H., Karasan, O.E., Pinar, M.C.: The robust spanning tree problem with interval data. Operations Research Letters **40**, 29–31 (2001)
18. Zieliński, P.: The computational complexity of the relative robust shortest path problem with interval data. European Journal of Operational Research **158**(3), 570–576 (2004)

MISTA 2019

Rescheduling for new orders on a single machine with rejection

Wenchang Luo · Kan Fang · Miaomiao Jin

Abstract We consider a single machine rescheduling problem where a set of original jobs has been scheduled to minimize the total weighted completion time. However, before formal processing begins, a new set of jobs arrives and creates a disruption. To reduce the negative impact of new jobs and keep an acceptable service level for the original jobs, the decision maker can either reject or outsource a subset of the new jobs by paying certain rejection penalties, and reschedule the original and the remaining new jobs without excessively disrupting the original schedule, which is measured by the maximum completion time deviation for any original jobs between the original and adjusted schedules. We examine a general model where the maximum completion time disruption appears both as a constraint and as a part of the cost objective. The overall cost objective is to minimize the sum of total weighted completion time of the original jobs and the accepted new jobs in the adjusted schedule, the weighted maximum completion time deviation and the total rejection cost. We first provide a dynamic programming-based exact algorithm running in pseudo-polynomial time, and then propose a fully polynomial time approximation scheme. Given the NP-hardness for the studied problem, our result is the best possible.

1 Introduction

In recent years, more and more researchers have started to consider various rescheduling problems arising from different dynamic production and service environments, such as automotive manufacturing [4], space shuttle mission [37], de-regulated power market [6], operating rooms [1] and [14], and so on. Among these problems, the common

Wenchang Luo
Scool of Mathematics and Statistics, Ningbo University, Ningbo 315211, China
E-mail: luowenchang@163.com

Kan Fang
College of Management and Economics, Tianjin University, Tianjin 300072, China
E-mail: kfang@tju.edu.cn

Miaomiao Jin
Scool of Mathematics and Statistics, Ningbo University, Ningbo 315211, China
E-mail: 1491190878@qq.com

types of disruptions creating the necessity for rescheduling include job cancellation [13], changes in order available times [10] and [28], machine breakdowns [16], [30] and [17], and so on. In particular, in the seminal work of Hall and Potts [9], the authors introduced a new type of rescheduling problem, in which a set of original jobs has already been scheduled for execution on a single machine. However, before formal processing begins, a new set of jobs arrives and creates a disruption. The decision-maker needs to reschedule the original and new jobs without excessively disrupting the original schedule. This kind of problem is referred to as a *Rescheduling problem for new orders*, and has attracted more and more attention from researchers during the past decade. We give a few examples. Hall et al. Yuan and Mu [11] further considered an extended version of the above problem with multiple new orders. Yuan and Mu [31] studied four different variants of rescheduling problems for new orders to minimize makespan, in which each job has a release date, and the corresponding disruption criterion is bounded. Yang [29] considered a rescheduling problem in which the processing times of newly arriving jobs can be reduced with a compression cost. Zhao and Tang [34] studied the rescheduling problem with linear deteriorating jobs on a single machine. Hoogeveen at al. [12] considered varous rescheduling problems for new orders in which switching processing from different job families incurs setup times, and the objective is to minimize the sum of setup times. Zhao and Yuan [35] considered the rescheduling problem with release dates to minimize the makespan and total sequence disruption simultaneously. For more pointers on the general rescheduling problems, we refer the reader to the surveys by Vieira et al. [26], Aytug et al. [2], and Ouelhadj and Petrovic [21].

On the other hand, due to the limited production capacity and tight delivery requirements, it may not be able to accept all the jobs for many highly loaded make-to-order production systems, otherwise both high inventory cost and remarkable customer dissatisfaction may be incurred due to the delay of some jobs. Thus, the manufacturer may wish to reject or outsource part of orders with certain penalties, so as to keep an acceptable service level for the accepted orders. As a result, a higher level decision concerning the partition of the jobs into accepted and rejected portions should be made prior to the scheduling decision. Because of its extensive applications in the make-to-order production systems, the *scheduling problem with job rejection* (or equivalently the *order acceptance and scheduling* problem) has been investigated by an enormous number of researchers, starting with the work by Bartal et al. [3], who studied the problem of minimizing the makespan of scheduling the accepted jobs plus the sum of penalties of the rejected jobs on identical parallel machines. To solve such problem, researchers have developed various algorithms and techniques with diverse machine characteristics and different objective functions. For example, Engels et al. [7] provided several techniques for designing algorithms to minimize the sum of the weighted completion time of the accpeted jobs and the sum of penalties of the rejected jobs. Zhang et al. [32] studied a scheduling problem with release dates and job rejection on a single machine, and the objective is to minimize the sum of the makespan of the accepted jobs and the total rejection penalty of the rejected jobs. Cheng and Sun [5] considered the problem of scheduling linear deteriorating jobs with rejection on a single machine to minimize some classical scheduling objectives and the total penalty of the rejected jobs. Zhang et al. [33] analyzed the computational complexities of the problems for scheduling jobs on a single machine under the job rejection constraint with distinct objective functions. Nobibon and Leus [18] studied a generalization of the order acceptance and scheduling problem in which the capacity available for processing the accepted orders is limited,

and devised different exact algorithms for the problem they proposed. Shabtay [22] considered a scheduling problem with rejection on a single serial batching machine to minimize the total completion time and the total rejection cost. Zhong et al. [36] considered an order acceptance and scheduling model with machine availability constraints to minimize the makespan of all accepted orders plus the total penalty of all rejected orders. Thevenin et al. [25] proposed different metaheuristics for a scheduling problem with rejection and tardiness penalties, in which sequence-dependent setup times will be performed if different families of jobs are processed successively. Ou et al. [20] developed an improved heuristic for parallel machine scheduling with rejection to minimize the completion time of the last accepted job plus the total penalty of all rejected jobs. Ou and Zhong [19] studied a bicriteria order acceptance and scheduling problem with consideration of fill rate in a parallel machine environment and presented efficient approximation algorithms. For more information on scheduling problems with rejection, we refer the reader to the review by Slotnick [24] and Shabtay et al. [23].

Inspired by the above two streams of research papers, it could be very interesting to integrate rescheduling for new orders and scheduling with job rejection into a unified model both from a practical and a theoretical point of view. However, according to our literature review, research on such topic appears to be rather sparse. The only three exceptions that we found are the following: Wang et al. [27] investigated a multi-objective dynamic scheduling in response to continuous arrivals of new jobs under the assumption that jobs can be rejected and the processing time of each job is dependent on its start time and can be controlled by allocating extra amount of resource; Wang et al. [28] studied a parallel machine rescheduling problem with job unavailability and rejection, and the objective is to minimize the total completion time, while the disruption measure and the total rejection cost are bounded by some given limits; Liu [15] considered a two-machine flow shop outsourcing and rescheduling problem in response to an unexpected arrival of new jobs, and proposed a hybrid variable neighborhood search algorithm to solve the problem.

In this work, we investigate a rescheduling problem for new orders with job rejection, in which some of the newly arrived jobs may be rejected by paying the rejection cost, and the disruption is measured as the maximum completion time deviation for any original job between the orginal and adjusted schedules. In particular, we consider the general model where the maximum completion time disruption appears both as a constraint and as a part of the cost objective. The overall cost objective is to minimize the sum of total weighted completion time of the original jobs and the accepted jobs from the new jobs in the adjusted schedule, the weighted maximum completion time deviation plus the total rejection cost.

The rest of the paper is organized as follows. In Section 2, we formally state the mathematical description of our scheduling model, followed by the development of a dynamic programming exact algorithm running in pseudo-polynomial time in Section 3. Then, we further propose a fully polynomial time approximation scheme for the problem in Section 4. Finally, we conclude our research and provide possible future research directions in Section 5.

2 Mathematical description of the problem

Following similar notations as in Hall and Potts [9], we formally describe our problem as follows: Let $\mathcal{J}_O = \{1, 2, \cdots, n_O\}$ denote a set of original non-preemptive jobs to be

processed on a single machine. We assume that these original jobs have been scheduled optimally to minimize the total weighted completion time by the well known *weighted shortest processing time first (WSPT)* rule, and that π^* is an optimal schedule with no idle time between the jobs. In view of this, a promised deliver time has been made for each job j in $\mathcal{J}_O$ according to its completion time in the schedule π^*. Let $\mathcal{J}_N = \{n_O + 1, n_O + 2, \cdots, n_O + n_N\}$ denote a set of new jobs that arrive at time zero and create a disruption after the planned schedule π^* for the jobs in $\mathcal{J}_O$ has been determined, but before formal processing begins. As mentioned by [9], we do not lose much by making the above assumption, this reason is as follows: if the jobs arrive after time zero, we can simply remove the fully processed jobs in $\mathcal{J}_O$ from the processing, wait those partially processed jobs until their completion time, and then we update $\mathcal{J}_O$ and n_O accordingly.

Let $\mathcal{J} = \mathcal{J}_O \cup \mathcal{J}_N$ and $n = n_O + n_N$. Let p_j and w_j denote the positive processing time and weight of job j for $j = 1, 2, \cdots, n$, respectively. We assume that all p_j and w_j are given as integers, and define $P_O = \sum_{j\in\mathcal{J}_O} p_j$. According to the planned schedule π^*, a promised deliver time $C_j(\pi^*)$ has been made for job j in $\mathcal{J}_O$, and we are asked to reschedule all the jobs in $\mathcal{J}_O$ and some jobs in $\mathcal{J}_N$ to minimize the total completion time subject to the promised deliver time for each job $j \in \mathcal{J}_O$ not violating too much. To be specific, for any feasible schedule σ of the jobs in $\mathcal{J}$, we define $C_j(\sigma)$ as the completion time of job j, and $\Delta_j(\pi^*, \sigma) = |C_j(\sigma) - C_j(\pi^*)|$ as the completion time disruption of job $j \in \mathcal{J}_O$. In addition, we define $\Delta_{\max}(\pi^*, \sigma) = \max_{j\in\mathcal{J}_O}\{\Delta_j(\pi^*, \sigma)\}$ as the maximum completion time disruption. When there is no ambiguity, we denote the above notations to C_j, Δ_j, and $\Delta_{\max}$, respectively.

To reduce the negative impact of the arrivals of new jobs and achieve an acceptable service level, we have the option to reject the processing of some jobs from $\mathcal{J}_N$ in rescheduling, where each rejected job may either be outsourced or not be processed at all. Naturally, rejecting a job $j \in \mathcal{J}_N$ incurs an additional rejection cost e_j due to either the outsourcing cost or the loss in income and customer goodwill. For simplicity, we denote the sets of accepted jobs and rejected jobs by $\mathcal{J}_N^A$ and $\mathcal{J}_N^{\bar{A}}$, respectively. Since there is no need to schedule the set of rejected job, we only need to schedule the jobs in $\mathcal{J}_O \cup \mathcal{J}_N^A$. In addition, the maximum completion time disruption models the penalties that result from changing of delivery times of jobs in $\mathcal{J}_O$ to customers, as well as the cost associated with the rescheduling of resources so that they are available when needed.

Using the standard three-field scheduling notation [8], we denote our problem by $1|rej, \Delta_{\max} \le k|\sum_{j\in\mathcal{J}_O\cup\mathcal{J}_N^A} w_jC_j + \mu\Delta_{\max} + \sum_{j\in\mathcal{J}_N^{\bar{A}}} e_j$, where we use $\Delta_{\max} \le k$ to indicate that the maximum completion time disruption is no greater than a constant integer k, which can be determined by managers to provide a maximum allowable disruption to the rescheduling process. In the objective function that we aim to minimize, we denote $\sum_{j\in\mathcal{J}_O\cup\mathcal{J}_N^A} w_jC_j$ as the total weighted completion time of the original and the accepted new jobs, $\sum_{j\in\mathcal{J}_N^{\bar{A}}} e_j$ as the total rejection cost and a constant $\mu \ge 0$ as the relative cost of one unit of disruption compared to the scheduling cost.

Note that when we set $\mathcal{J}_O = \emptyset$, then our problem becomes the problem of $1|rej|\sum_{j\in\mathcal{J}_N^A} w_jC_j + \sum_{j\in\mathcal{J}_N^{\bar{A}}} e_j$, which has been shown to be NP-hard by Engels et al. [7]. Therefore, the problem we considered in this paper is also NP-hard. In particular, our problem encompasses the following three special cases:

1. the ordinary scheduling problem with job rejection when we both set k to be sufficiently large and $\mu = 0$, i.e.,

$$1|rej| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$$

2. the constrained scheduling problem with job rejection when we set $\mu = 0$, i.e.,

$$1|rej, \Delta_{\max} \leq k| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$$

3. the total cost problem when we set k to be sufficiently large, i.e.,

$$1|rej| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \mu \Delta_{\max} + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$$

3 A dynamic programming-based exact algorithm

In this section, we first present an important structural property on the optimal schedules that will be used later in the design of our proposed dynamic programming-based exact algorithm.

Lemma 1 *For the problem* $1|rej, \Delta_{\max} \leq k| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \mu \Delta_{\max} + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$, *there exists an optimal schedule* σ^* *in which*

- *the accepted jobs from* $\mathcal{J}_N$ *that scheduled before the last job in* $\mathcal{J}_O$ *plus all the jobs in* $\mathcal{J}_O$ *follow the WSPT order.*
- *the accepted jobs from* $\mathcal{J}_N$ *that scheduled after the last job in* $\mathcal{J}_O$ *also follow the WSPT order.*

Based on the structural property we obtained above, we are now able to derive our dynamic programming-based exact algorithm as follows: We first sort all the jobs in $\mathcal{J}$ according to the WSPT rule, without loss of generality, we assume that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$. To describe more conveniently, we define $p_{\max} = \max_{j \in \mathcal{J}} p_j$, $w_{\max} = \max_{j \in \mathcal{J}} w_j$, and $e_{\max} = \max_{j \in \mathcal{J}_N} e_j$. In addition, we refer to the partial schedule of jobs that scheduled before job j_o (including j_o) as the *left schedule* and the one that are scheduled after job j_o as the *right schedule*.

Then, we use a six-dimensional vector $(j; C_j, \ell_j, Z_j, \Delta_j, E_j)$ to represent a state, which is true if and only if there exists a partial schedule on the first j jobs, such that the current completion time is C_j in the left schedule, the current total processing time of jobs in $\mathcal{J}_N$ is ℓ_j, the current total weighted completion time is Z_j, the current maximum completion time disruption is Δ_j, and the current total rejection cost is E_j. Note that the initial state $(0; C_0, \ell_0, Z_0, \Delta_0, E_0) = (0; 0, 0, 0, 0, 0)$ is obviously true. Using the above definition, we can now establish the recurrence relationship between the true states in which the first element of the six-dimensional vector changes from j to $j + 1$, as shown in Theorem 1.

Theorem 1 *Starting from a true state* $(j; C_j, \ell_j, Z_j, \Delta_j, E_j)$, *the possible true states of the form* $(j + 1; C_{j+1}, \ell_{j+1}, Z_{j+1}, \Delta_{j+1}, E_{j+1})$ *are given as follows:*

- *if job* $j+1 \in \mathcal{J}_O$*, then the true state is* $(j+1; C_j + p_{j+1}, \ell_j, Z_j + w_{j+1}(C_j + p_{j+1}), \Delta_j, E_j)$*;*
- *if job* $j+1 \in \mathcal{J}_N$ *is accepted with* $j+1 < j_o$*, then the true state is* $(j+1; C_j + p_{j+1}, \ell_j + p_{j+1}, Z_j + w_{j+1}(C_j + p_{j+1}), \Delta_j + p_{j+1}, E_j)$*;*
- *if job* $j+1 \in \mathcal{J}_N$ *is accepted with* $j+1 > j_o$*, then the true state is* $(j+1; C_j, \ell_j + p_{j+1}, Z_j + w_{j+1}(P_O + \ell_{j+1}), \Delta_j, E_j)$*;*
- *if job* $j+1 \in \mathcal{J}_N$ *is rejected, then the true state is* $(j+1; C_j, \ell_j, Z_j, \Delta_j, E_j + e_j)$.

Given the above recurrences as shown in Theorem 1, starting from the initial true state $(0;0,0,0,0,0)$, we are able to calculate all the true states of the form $(j; C, \ell, Z, \Delta, E)$, where $0 \le j \le n$, $0 \le C \le np_{\max}$, $0 \le \ell \le n_N p_{\max}$, $0 \le Z \le n^2 w_{\max} p_{\max}$, $0 \le \Delta \le k$, and $0 \le E \le n_N e_{\max}$. For simplicity, we denote $I^C = [0, np_{\max}], I^\ell = [0, n_N p_{\max}], I^Z = [0, n^2 w_{\max} p_{\max}], I^\Delta = [0,k], I^E = [0, n_N e_{\max}]$. Putting this all together, we can solve the problem using Algorithm 3.1 described below. For simplicity, we denote this algorithm as Algorithm DP.

Algorithm 3.1 A dynamic programming-based exact algorithm.

1: Sort all the jobs in $\mathcal{J}_O \cup \mathcal{J}_N$ in nonincreasing order of w_j/p_j, i.e., $w_1/p_1 \ge w_2/p_2 \ge \ldots \ge w_n/p_n$. Denote the last job belonging to $\mathcal{J}_O$ as j_o.

2: Starting from the initial true state (0;0,0,0,0,0), determine all the true states of the form $(j; C, \ell, Z, \Delta, E)$ using the recurrences as shown in Theorem 1, where $0 \le j \le n$, $0 \le C \le np_{\max}$, $0 \le \ell \le n_N p_{\max}$, $0 \le Z \le n^2 w_{\max} p_{\max}$, $0 \le \Delta \le k$, and $0 \le E \le n_N e_{\max}$.

3: Among all the true states of the form $(n; C_n, \ell_n, Z_n, \Delta_n, E_n)$, find out the one with minimum value of $Z_n + \mu\Delta_n + E_n$ subject to $\Delta_n \le k$. Determine the corresponding subset of accepted jobs from $\mathcal{J}_N$ before j_o, and the subset of accepted jobs from $\mathcal{J}_N$ after j_o.

4: Schedule the above jobs according to the rules as described in Lemma 1.

Theorem 2 *Algorithm DP finds an optimal solution for problem* $1|rej, \Delta_{\max} \le k| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \mu\Delta_{\max} + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$ *in* $O(n^6 k p_{\max}^3 w_{\max} e_{\max})$ *time.*

4 A fully polynomial time approximation scheme

In this section, we transform our proposed dynamic programming-based exact algorithm, which runs in pseudo-polynomial time, into a fully polynomial time approximation scheme (FPTAS). Given any ε with $\varepsilon > 0$, we define $\delta = (1 + \varepsilon/2n)$ and set $v_1 = \lceil \log_\delta^{np_{\max}} \rceil$, $v_2 = \lceil \log_\delta^{n_N p_{\max}} \rceil$, $v_3 = \lceil \log_\delta^{n^2 w_{\max} p_{\max}} \rceil$, $v_4 = \lceil \log_\delta^{k} \rceil$ and $v_5 = \lceil \log_\delta^{n_N e_{\max}} \rceil$. In addition, for any positive integer n, we denote $[n]$ as the set of $\{0, 1, 2, \ldots, n\}$.

We first partition the sets of $I^C, I^\ell, I^Z, I^\Delta$, and I^E as follows:

- Partition of interval I^C: $I_0^C = [0,1], I_i^C = [\delta^{i-1}, \delta^i], 1 \le i \le v_1 - 1, I_{v_1}^C = [\delta^{v_1 - 1}, np_{\max}]$;
- Partition of interval I^ℓ: $I_0^\ell = [0,1], I_i^\ell = [\delta^{i-1}, \delta^i], 1 \le i \le v_2 - 1, I_{v_2}^\ell = [\delta^{v_2 - 1}, n_N p_{\max}]$;
- Partition of interval I^Z: $I_0^Z = [0,1], I_i^Z = [\delta^{i-1}, \delta^i], 1 \le i \le v_3 - 1, I_{v_3}^Z = [\delta^{v_3 - 1}, n^2 w_{\max} p_{\max}]$;
- Partition of interval I^Δ: $I_0^\Delta = [0,1], I_i^\Delta = [\delta^{i-1}, \delta^i], 1 \le i \le v_4 - 1, I_{v_4}^\Delta = [\delta^{v_4 - 1}, k]$;

– Partition of interval I^E: $I_0^E = [0,1], I_i^E = [\delta^{i-1}, \delta^i], 1 \le i \le v_5 - 1, I_{v_5}^E = [\delta^{v_5-1}, n_N e_{\max}]$.

This results in the partition of $I = I^C \times I^\ell \times I^Z \times I^\Delta \times I^E$ into a set of $v_1 \times v_2 \times v_3 \times v_4 \times v_5$ five-dimensional subintervals.

Then, for each element $(q_1, q_2, q_3, q_4, q_5) \in [v_1] \times [v_2] \times [v_3] \times [v_4] \times [v_5]$, we define a box set $S(j; q_1, q_2, q_3, q_4, q_5)$ as the collection of the states $(j; C_j, \ell_j, Z_j, \Delta_j, E_j)$ such that $C_j \in I_{q_1}^C$, $\ell_j \in I_{q_2}^\ell$, $Z_j \in I_{q_3}^Z$,$\Delta_j \in I_{q_4}^\Delta$, and $E_j \in I_{q_5}^E$. We also define a boolean entry $B(j; q_1, q_2, q_3, q_4, q_5)$, which is true if and only if there exists a partial schedule on the first j jobs, such that the corresponding state $(j; C_j, \ell_j, Z_j, \Delta_j, E_j) \in S(j; q_1, q_2, q_3, q_4, q_5)$. In addition, we associate each boolean entry with the partial schedule that with the least value of Δ_j, that is, all other partial schedules which also evaluate this boolean entry to be true are discarded, ties are broken arbitrarily. For initialization, $B(0; 0, 0, 0, 0, 0)$ is obviously true since we can find an associated partial schedule with state $(0; 0, 0, 0, 0, 0)$, which belongs to the set $S(0; 0, 0, 0, 0, 0)$. Note that all other boolean entries $B(0; q_1, q_2, q_3, q_4, q_5)$ are false for any $(q_1, q_2, q_3, q_4, q_5) \neq (0, 0, 0, 0, 0)$.

With the above definitions, we can now present our fully polynomial time approximation scheme for the problem of $1|rej, \Delta_{\max} \le k| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \mu \Delta_{\max} + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$ as in Algorithm 4.1. For simplicity, we denote this algorithm as Algorithm A_ε.

Algorithm 4.1 A fully polynomial time approximation scheme.

1: Initialization: $B(0; 0, 0, 0, 0, 0)$ is true, and the corresponding state is $(0; 0, 0, 0, 0, 0)$.

2: for $j = 0$ to $n - 1$

3: for any state $s = (j; C_j, \ell_j, Z_j, \Delta_j, E_j)$ such that $B(j; q_1, q_2, q_3, q_4, q_5)$ is true for some $(q_1, q_2, q_3, q_4, q_5) \in [v_1] \times [v_2] \times [v_3] \times [v_4] \times [v_5]$

4: Use the recurrences (i.e., cases (a)-(d) in Theorem 1) to calculate the possible true state $\tilde{s} = (j + 1; C_{j+1}, \ell_{j+1}, Z_{j+1}, \Delta_{j+1}, E_{j+1})$, determine the corresponding vector $(p_1, p_2, p_3, p_4, p_5)$ such that $\tilde{s} \in S(j + 1; p_1, p_2, p_3, p_4, p_5)$.

5: if $B(j + 1; p_1, p_2, p_3, p_4, p_5)$ is false

6: Set $\tilde{s}$ as the associated state of $B(j + 1; p_1, p_2, p_3, p_4, p_5)$. Denote the state s associated with $B(j; q_1, q_2, q_3, q_4, q_5)$ as the predecessor of the state $\tilde{s}$ associated with $B(j + 1; p_1, p_2, p_3, p_4, p_5)$.

7: else

8: Suppose its previous associated state is $s' = (j+1; C'_{j+1}, \ell'_{j+1}, Z'_{j+1}, \Delta'_{j+1}, E'_{j+1})$.

9: Compare $\tilde{s}$ and s', choose the one with smaller value of Δ and update it as the associated state of $B(j + 1; p_1, p_2, p_3, p_4, p_5)$. Update the predecessor relationship accordingly.

10: for $(q_1, q_2, q_3, q_4, q_5) \in [v_1] \times [v_2] \times [v_3] \times [v_4] \times [v_5]$ such that $B(n; q_1, q_2, q_3, q_4, q_5)$ is true

11: Choose the state that with minimum value of $Z_n + \mu \Delta_n + E_n$ with $\Delta_n \le k$ and denote this state as $(n; C_n^\varepsilon, \ell_n^\varepsilon, Z_n^\varepsilon, \Delta_n^\varepsilon, E_n^\varepsilon)$.

Theorem 3 *For any $\varepsilon > 0$, Algorithm A_ε is an FPTAS for the problem of $1|rej, \Delta_{\max} \le k| \sum_{j \in \mathcal{J}_O \cup \mathcal{J}_N^A} w_j C_j + \mu \Delta_{\max} + \sum_{j \in \mathcal{J}_N^{\bar{A}}} e_j$, with a running time of $O(n^6 (\log M)^5 / \varepsilon^5)$, where $M = \max\{k, p_{\max}, w_{\max}, e_{\max}\}$.*

5 Conclusions

In this paper, we focus on the issue of rescheduling for new orders on a single machine with rejection. The objective is to minimize the sum of total weighted completion time of the original jobs and the accepted jobs from the new jobs, the weighted maximum completion time deviation and the total rejection cost, under the constraint that the maximum completion time deviation is less than a given value. We derive a pseudo-polynomial time dynamic programming-based exact algorithm, and then present a fully polynomial time approximation scheme for this problem. Given the NP-hardness of the problem, our result achieves the best possible from the approximation algorithm perspective. It would be interesting to extend our model to different machine environments, e.g., the flow shop and parallel machines.

Acknowledgements Luo was supported by the Natural Science Foundation of China (Grant No. 11971252), Humanities and Social Sciences Planning Foundation of the Ministry of Education (Grant No. 18YJA630077), Zhejiang Provincial Natural Science Foundation of China(Grant No. LY19A010005), and Ningbo Natural Science Foundation (Grant No. 2018A610198). Fang was supported by the Natural Science Foundation of China (Grant No. 71701144, 71528002, 91646118).

References

1. B. Akbarzadeh, G. Moslehi, M. Reisi-Nafchi, B. Maenhout, The re-planning and scheduling of surgical cases in the operating room department after block release time with resource rescheduling, European Journal of Operational Research, 278(2), 596–614 (2019)
2. H. Aytug, M. A. Lawley, K. McKay, S. Mohan, R. Uzsoy, Executing production schedules in the face of uncertainties: A review and some future directions, European Journal of Operational Research, 161(1), 86–110 (2005)
3. Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie, Multiprocessor scheduling with rejection, SIAM Journal on Discrete Mathematics, 13(1), 64–78 (2000)
4. J. C. Bean, J. R. Birge, J. Mittenthal, C. E. Noon, Matchup scheduling with multiple resources, release dates and disruptions, Operations Research, 39(3), 470–483 (1991)
5. Y. Cheng, S. Sun, Scheduling linear deteriorating jobs with rejection on a single machine, European Journal of Operational Research, 194(1), 18–27 (2009)
6. K. Dahal, K. Al-Arfaj, K. Paudyal, Modelling generator maintenance scheduling costs in deregulated power markets, European Journal of Operational Research, 240(2), 551–561 (2015)
7. D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. Uma, J. Wein, Techniques for scheduling with rejection, Journal of Algorithms, 49(1), 175–191 (2003)
8. R. L. Graham, E. L. Lawler, J. K. Lenstra, A.H.G R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of discrete mathematics, 5, 287–326 (1979)
9. N. G. Hall, C. N. Potts, Rescheduling for new orders. Operations Research, 52(3), 440–453 (2004)
10. N. G. Hall, C. N. Potts, Rescheduling for job unavailability, Operations research, 58(3), 746–755 (2010)
11. N. G. Hall, Z. Liu, C. N. Potts, Rescheduling for multiple new orders, INFORMS Journal on Computing, 19(4), 633–645 (2007)
12. H. Hoogeveen, C. Lenté, V. Tkindt, Rescheduling for new orders on a single machine with setup times, European Journal of Operational Research, 223(1), 40–46 (2012)
13. A.K. Jain, H.A. Elmaraghy, Production scheduling/rescheduling in exible manufacturing, International Journal of Production Research, 35(1), 281–309 (1997)
14. K. S. Jung, M. Pinedo, C. Sriskandarajah, V. Tiwari, Scheduling elective surgeries with emergency patients at shared operating rooms, Production and Operations Management, 28(6), 1407–1430 (2019)

15. L. Liu, Outsourcing and rescheduling for a two-machine flow shop with the disruption of new arriving jobs: A hybrid variable neighborhood search algorithm, Computers & Industrial Engineering, 130, 198–221 (2019)
16. Z. Liu, Y. K. Ro, Rescheduling for machine disruption to minimize makespan and maximum lateness. Journal of Scheduling, 17(4), 339–352 (2014)
17. W. Luo, T. Luo, R. Goebel, G. Lin, Rescheduling due to machine disruption to minimize the total weighted completion time, Journal of Scheduling, 21(5), 565–578 (2018)
18. F. T. Nobibon, R. Leus, Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment, Computers & Operations Research, 38(1), 367–378 (2011)
19. J. Ou, X. Zhong. Bicriteria order acceptance and scheduling with consideration of fill rate, European Journal of Operational Research, 262(3), 904–907 (2017)
20. J. Ou, X. Zhong, G. Wang, An improved heuristic for parallel machine scheduling with rejection, European Journal of Operational Research, 241(3), 653–661 (2015)
21. D. Ouelhadj, S. Petrovic, A survey of dynamic scheduling in manufacturing systems, Journal of scheduling, 12(4), 417–431 (2009)
22. D. Shabtay, The single machine serial batch scheduling problem with rejection to minimize total completion time and total rejection cost, European Journal of Operational Research, 233(1), 64–74 (2014)
23. D. Shabtay, N. Gaspar, M. Kaspi, A survey on offline scheduling with rejection, Journal of Scheduling, 16(1), 3–28 (2013)
24. S. A. Slotnick, Order acceptance and scheduling: A taxonomy and review, European Journal of Operational Research, 212(1), 1–11 (2011)
25. S. Thevenin, N. Zufferey, M. Widmer, Metaheuristics for a scheduling problem with rejection and tardiness penalties, Journal of Scheduling, 18(1), 89–105 (2015)
26. G. E. Vieira, J. W. Herrmann, E. Lin, Rescheduling manufacturing systems: a framework of strategies, policies, and methods, Journal of Scheduling, 6(1), 39–62 (2003)
27. D. Wang, F. Liu, Y. Jin, A multi-objective evolutionary algorithm guided by directed search for dynamic scheduling. Computers & Operations Research, 79, 279–290 (2017)
28. D. Wang, Y. Yin, T.C.E. Cheng, Parallel-machine rescheduling with job unavailability and rejection, Omega, 81, 246–260 (2018)
29. B. Yang, Single machine rescheduling with new jobs arrivals and processing time compression, The International Journal of Advanced Manufacturing Technology, 34(3-4), 378–384 (2007)
30. Y. Yin, T.C.E. Cheng, D. Wang, Rescheduling on identical parallel machines with machine disruptions to minimize total completion time, European Journal of Operational Research, 252(3), 737–749 (2016)
31. J. Yuan, Y. Mu, Rescheduling with release dates to minimize makespan under a limit on the maximum sequence disruption, European Journal of Operational Research, 182(2), 936–944 (2007)
32. L. Zhang, L. Lu, J. Yuan, Single machine scheduling with release dates and rejection, European Journal of Operational Research, 198(3), 975–978 (2009)
33. L. Zhang, L. Lu, J. Yuan, Single-machine scheduling under the job rejection constraint, Theoretical Computer Science, 411(16-18), 1877-1882 (2010)
34. C. Zhao, H. Tang, Rescheduling problems with deteriorating jobs under disruptions, Applied Mathematical Modelling, 34(1), 238–243 (2010)
35. Q. Zhao, J. Yuan, Pareto optimization of rescheduling with release dates to minimize makespan and total sequence disruption, Journal of Scheduling, 16(3), 253–260 (2013)
36. X. Zhong, J. Ou, G. Wang, Order acceptance and scheduling with machine availability constraints, European journal of operational research, 232(3), 435–441 (2014)
37. M. Zweben, E. Davis, B. Daun, M. J. Deale, Scheduling and rescheduling with iterative repair, IEEE Transactions on Systems, Man, and Cybernetics, 23(6), 1588–1596 (1993)

MISTA 2019

Two rescheduling problems with allowing for the unexpected arrival of new jobs

Xingong Zhang[a], Daiyu Bai[b],

Win-Chin Lin[c], Chin-Chia Wu[c*]

a. College of Mathematics Science, Chongqing Normal University,

Chongqing, China

b. School of Maritime Economics & Management, Dalian Maritime University,

Dalian, 116026, China

c. Department of Statistics, Feng Chia University,

Taichung, Taiwan

Abstract. In modern manufacturing and services, rescheduling frequently happens in manufacturing practice that a disruption creates a need. This paper studies the issue of rescheduling to allow for the unexpected arrival of new jobs, taking into account the effect of time disruptions on a previously planned optimal schedule. Rescheduling means that a set of original jobs has already been scheduled to minimize some classical objective, then a new set of jobs arrives and creates a disruption. We consider the rescheduling problems with maximum weighted tardiness costs under a limit of time disruptions from the original schedule. Two time disruption problems will be involved: under an upper restriction on the total time disruption and relaxing time disruption restriction. For the former case, We give the proof of strongly NP-hard by reducing 3-Partition problem. For latter case, we present strong NP-hard proof and provide a 2-approximation polynomial time algorithm, and show that, unless $P = NP$, the proposed problem can not have an approximation polynomial time algorithm with a performance ratio less than 2. One of the proposed algorithms is improved by three local searches, respectively, as

*E-mail addresses: cchwu@fcu.edu.tw

three seeds used in simulated annealing for approximate solution. Finally, we also present some extensive numerical studies to evaluate the performance of these algorithms.

Key words: Rescheduling; Single-machine;Time disruption; the Maximum weighted tardiness; Simulated annealing.

1 Introduction

Our research is motivated by the strong application background of rescheduling problems. Rescheduling has been studied extensively in the field of scheduling theory in the last decade. It frequently happens in manufacturing practice that a disruption creates a need for rescheduling. Such as, Foxconn Technology Group is the world's largest provider of electronics manufacturing services. Notable products manufactured by Foxconn include iPhone devices, Nokia devices, Huawei devices, Xiaomi devices, PlayStation 3, et al. Foxconn factories manufactured an estimated 40% of all consumer electronics sold worldwide. Though Foxconn has performed produce schedules of products, it will have to adjust the proposed produce schedules to accept new orders from major accounts due to unexpected events. As we all know, Huawei devices have to switch from Flex to Foxconn to make P30 phone. To obtain more revenue, Foxconn needs to adjust the existing schedule to compensate disruption, with a view to obtaining a cost efficient schedule without causing too much inconveniences to other customers. Examples of common disruptions include: the arrival of new orders, order cancelations, changes in order priority, processing delays, changes in release dates, machine breakdowns, and the unavailability of raw materials, personnel, or tools. Rescheduling involves adjusting a previously planned, possibly optimal, scheduling to account for a disruption. Two extensive reviews of rescheduling literature involving disruptions or uncertainties can be found in Aytug et al.[2] and Vieira et al.[19] until the begin of 21 century.

Problem formulation: This paper considers single-machine rescheduling problems with maximum weighted tardiness. By the notation of Hall and Potts[6], the problem can be formally stated as follows: Let $J_0 = \{J_1, \cdots, J_{n_0}\}$ denote a set of original jobs to be processed non-preemptively on a single machine. Assume that these jobs have been scheduled optimally to minimize some classical objective and that π^* is an optimal schedule with no idle time between the jobs. Let $J_N = \{J_{n_0+1}, \cdots, J_{n_0+n_N}\}$ denote a set of new jobs that arrive together. Furthermore, Assume that these jobs arrive at time zero after a schedule for the jobs of J_0 has been determined, but before processing begins. There is no loss of generality in this assumption: If the jobs arrive after time zero, then the fully processed jobs of J_0 are removed from the problem, any partly processed jobs are processed to completion, and J_0 and n_0 are updated accordingly. Let $J = J_0 \bigcup J_N$ and $n = n_0 + n_N$. Each job $J_j \in J$ has an integral normal processing time p_j, a weight $w_j > 0$ and due date d_j. For any schedule σ of the jobs in J, the following variables will be defined:

$S_j(\sigma)$ is the time at which job $J_j \in J$ starts its processing in schedule σ.
$C_j(\sigma)$ is the time at which job $J_j \in J$ is completed in schedule σ.
$T_j(\sigma) = \max\{C_j(\sigma) - d_j, 0\}$ is the tardiness value of job $J_j \in J$.
$\Delta_j(\pi^*; \sigma) = |C_j(\sigma) - C_j(\pi^*)|$ is the time disruption of job $J_j \in J_0$.
When there is no ambiguity, we simplify S_j, C_j, T_j, and $\Delta_j(\pi^*)$, respectively.

$\Delta_j(\pi^*) \leq k_j$, the time disruption of job J_j cannot exceed $k_j(\geq 0)$, where $k_j \geq 0$ is a known integer.

$\sum \Delta_j(\pi^*) \leq k$: $\sum_{J_j \in J_0} \Delta_j(\pi^*) \leq k$, the total time disruption of the jobs cannot exceed k, where $k \geq 0$ is a known integer.

Our contribution: For an upper restriction on the total time disruption, we give the proof of strongly NP-hard by reducing 3-Partition problem. For relaxing time disruption restriction, we present strong NP-hard proof and show that, unless $P = NP$, the proposed problem can not have an approximation polynomial time algorithm with a performance ratio less than 2. Finally, we also present simulation annealing algorithm studies to evaluate the performance of these algorithms, where algorithm $Ag2p_SA$ does very well among seven algorithms in term of the average error percentage (AEPs) and the relative percentages of deviation (RPDs)

Organization of this paper: This paper proposes two rescheduling problems with allowing for the unexpected arrival of new jobs. We consider the rescheduling problems to minimize the total tardiness costs under two limits of the disruption from the original scheduling, and present three simulated annealing methods for approximate solution. The rest of the paper is organized as follows. In the next section, we give the problem description. In Section 3 and 4, we consider two rescheduling problems with two time disruption constraints. Numerical experiment will be listed by three variants simulated annealing in Sec. 5. The last section is the conclusion.

2 Relevant research

The research in this paper is mainly related to the rescheduling model for new orders firstly introduced by Hall and Potts [6]. Rescheduling problem is described as follows: A set of original jobs has been scheduled (but has not been processed) optimally to minimize some cost function and promises have been made to customers based on the schedule,then a new set of jobs arrives unexpectedly before the processing starts. Rescheduling problem with newly arrived jobs are condiered on a single-machine to minimize the maximum lateness and the total completion time under a limit of the disruption from the original scheduling. Yang[23] studies the single-machine rescheduling with new jobs arrivals and processing time compression. Yuan and Mu[24] consider the rescheduling problem for jobs on a single machine with release dates to minimize makespan under a limit on the maximum sequence disruption. Zhao and Yuan[27] showed that minimizing the makespan and the total sequence disruption can be solvable in polynomial time to obtain the Pareto optimization rescheduling on a single machine. Zhao and Tang[26] studied linear deteriorating jobs jobs under some rescheduling problems for new orders and disruptions. For any limit of the disruption, the polynomial algorithm is proposed. Liu and Zhou[12] studied rescheduling for new orders with deterioration and learning effects under the maximum sequence disruption. Katragjini et al. [10] studied the flow shop rescheduling under different types of disruption and presented rescheduling algorithms to seek a good trade-off between schedule quality and stability. Zhao et al.[25] studied the rescheduling with new orders on a single machine under the general maximum allowable time disruptions. The objectives are to minimize the maximum lateness and the total completion time, respectively. Teghem and Tuyttens[17] studied bi-objective rescheduling problems to minimize the total completion time or the maximum tardiness. Algorithms are provided for the eight possible bi-objective problems and illustrated by some didactic examples. Luo et al.[14] investigated an unexpected machine unavailability under rescheduling problem. The objective function is to minimize the sum of the

total weighted completion time and the weighted maximum time deviation, under the limit of the constraint that the maximum time deviation. They also presented a pseudo-polynomial time exact algorithm and a fully polynomial time approximation scheme by dynamic programming technology. More recent papers with rescheduling and different machine environments can be referred as follows: Hoogeveen et al.[8], Kang and Zhang[9], Valledora et al.[18] and Yang [22].

Simulated annealing (SA), proposed by Kirkpatrick et al. [11], is one of the most popular meta-heuristic methods widely applied to solve combinatorial optimization problems. This approach has the advantage of avoiding getting trapped in a local optimum. This is due to the hill climbing moves, which are governed by a control parameter. Recently, simulated annealing methods have been involved in scheduling problems. Wu et al.[20] proposed three meta-heuristics of two-stage assembly linear deterioration scheduling problems, including dynamic differential evolution (DDE), simulated annealing (SA) algorithm, and cloud theory-based simulated annealing (CSA) algorithm for find near-optimal solutions. More recently scheduling problems with simulated annealing algorithm may be referred as follows: Akram and Kamal[1], Hao et al.[7], Moschakis and Karatza[15], Taghi Assadi and Bagheri[16] and Xiao et al.[21].

3 Notation and terminology

In this paper, we will consider the maximum weighted tardiness problem under total time disruption and different time disruption. Note that it is well known that problem $1||\sum w_jT_j$ can be solved by the algorithm developed by Lawler[13]. However, there are different between the sum of weighted tardiness problem and the maximum tardiness problem. So we next present a lemma to apply the following sections to solve the maximum tardiness problem.

Lemma 1 For problem $1|d_j = d|\max w_jT_j$, an optimal sequence will be obtained by non-increasing order of weight w_j.

Proof: Let jobs be not scheduled by non-increasing order of weights in an optimal sequence π, Furthermore, we assume that job J_i and job J_j are scheduled in position r and $r+1$, which the job J_i and J_j satisfies $w_i \leq w_j$. by interchanging the two jobs, we can obtain the new job sequence π', i.e., job J_i and job J_j are scheduled in position $r+1$ and r, respectively. Then we have

$$w_jT_j(\pi') = w_j\max(C_j(\pi') - d) \leq w_j\max(C_j(\pi) - d) = w_jT_j(\pi),$$
$$w_iT_i(\pi') = w_i\max(C_i(\pi') - d) \leq w_j\max(C_j(\pi) - d) = w_jT_j(\pi).$$

However, the tardiness of other jobs can not be changed. Thus, repeating this job interchange argument for all the jobs not sequenced in the LW (largest weight first) rule completes the proof. Then we have $\max w_jT_j(\pi') \leq \max w_jT_j(\pi)$. □

Next, We will consider the optimal sequence of the original jobs to illustrate the property of problem $1|\sum \Delta_j(\pi) \leq k|\max w_jT_j$. So we will present the following algorithm under weight $w_j > 0$ for $J_j \in J_0$.

Algorithm 1

Step 1: Let job set be $\Gamma = \{J_1, J_2, \cdots, J_n\}$ and the original job set be $\Phi = \emptyset$;

Step 2: Compute $(\sum_{i=1}^{n} p_i - d_j)w_j$, $J_j \in \Gamma$. Let J_k is one of minimizing the value $(\sum_{i=1}^{n} p_i - d_j)w_j$.

Step 3: Schedule J_k before sequence Φ to obtain a new schedule, i.e., $\Phi := \{J_k, \Phi\}$;

Step 4: Set $\Gamma = \Gamma - \{J_k\}$. If $\Gamma \neq \emptyset$, then turn to Step 2. Otherwise, output sequence Φ.

Theorem 2 For problem $1||\max w_j T_j$, an optimal sequence can be obtained in time $O(n^2 logn)$ by Algorithm 1.

Proof Let σ is job sequence obtained by algorithm 1, and σ' is any sequence, where jobs in sequence σ and sequence σ' are scheduled by different orders. Assume that J_j is the last job broken algorithm in sequence σ', and job J_i in σ is scheduled the same position with J_j in sequence σ', where $(\sum_{i=1}^{n} p_i - d_i)w_i < (\sum_{i=1}^{n} p_i - d_j)w_j$. Then we will interchange jobs J_i and J_j in sequence σ', and other jobs will hold the same position, so that a new sequence σ'' is obtained. Next, we will discuss the following cases:

- Job J_j in sequence σ' and Job J_i in in sequence σ''. If the two jobs are tardy, i.e., $w_j T_j(\sigma') = (C_j(\sigma') - d_j)w_j$ and $w_i T_i(\sigma'') = (C_i(\sigma'') - d_i)w_i$. Note that $C_j(\sigma') = C_i(\sigma'')$ and $(\sum_{i=1}^{n} p_i - d_i)w_i < (\sum_{i=1}^{n} p_i - d_j)w_j$, then we have $w_j T_j(\sigma') > w_i T_i(\sigma'')$; If the two jobs are early, i.e., $w_j T_j(\sigma') = w_i T_i(\sigma'') = 0$, then we have $w_j T_j(\sigma') = w_i T_i(\sigma'')$; If one of two jobs is tardy and another is early, i.e., $w_j T_j(\sigma') = (C_j(\sigma') - d_j)w_j$ and $w_i T_i(\sigma'') = 0$, then we have $w_j T_j(\sigma') > w_i T_i(\sigma'')$.

- Job J_i in sequences σ' and σ''. If the two jobs are tardy, i.e., $w_i T_i(\sigma') = (C_i(\sigma') - d_i)w_i$ and $w_i T_i(\sigma'') = (C_i(\sigma'') - d_i)w_i$. Note that $C_i(\sigma') < C_i(\sigma'')$, then we have $w_i T_i(\sigma') > w_i T_i(\sigma'')$; If the two jobs are early, i.e., $w_j T_j(\sigma') = w_i T_i(\sigma'') = 0$, then we have $w_i T_i(\sigma') = w_i T_i(\sigma'')$; If one of two jobs is tardy and another is early, i.e., $w_i T_i(\sigma') = (C_i(\sigma') - d_i)w_j$ and $w_i T_i(\sigma'') = 0$, then we have $w_i T_i(\sigma') > w_i T_i(\sigma'')$.

- Job J_j in sequences σ' and σ''. If the two jobs are tardy, i.e., $w_j T_j(\sigma') = (C_j(\sigma') - d_j)w_j$ and $w_j T_j(\sigma'') = (C_j(\sigma'') - d_j)w_j$. Note that $C_j(\sigma') > C_j(\sigma'')$, then we have $w_j T_j(\sigma') > w_j T_j(\sigma'')$; If the two jobs are early, i.e., $w_j T_j(\sigma') = w_j T_j(\sigma'') = 0$, then we have $w_j T_j(\sigma') = w_j T_j(\sigma'')$; If one of two jobs is tardy and another is early, i.e., $w_j T_j(\sigma') = (C_j(\sigma') - d_j)w_j$ and $w_j T_j(\sigma'') = 0$, then we have $w_j T_j(\sigma') > w_j T_j(\sigma'')$.

Furthermore, we can obtain $\max\{w_i T_i(\sigma''), w_j T_j(\sigma'')\} \leq \max\{w_i T_i(\sigma'), w_j T_j(\sigma')\}$. Note that the objective value of other jobs remains unchanged. It also implies that the objective function value of sequence σ'' cannot be increased by interchange job J_i and J_j. Hence, σ'' dominates σ'.

Next, we consider the complexity of Algorithm. Lemma 2 manifests the correction of Algorithm 1. Step 2 of Algorithm 1 takes $O(nlogn)$ time to obtain non-increasing order, and Step 3 will be performed at most $O(n)$ time. Moreover, the time complexity of Algorithm 1 is $O(n^2 logn)$ time. □

4 Total time disruption constraint

This section considers rescheduling problem with total time disruption constraint. we will present strongly NP-hardiness proof by reducing 3-Partition problem.

Theorem 1. For problem $1|\sum \Delta_j(\pi) \leq k|\max w_j T_j$ is strongly NP-hard.

Proof: We prove problem $1|\sum \Delta_j(\pi) \leq k|\max w_j T_j$ by reducing the strongly NP-complete 3-Partition problem [5].

3-Partition problem: Given $3t+1$ positive integers $a_1, a_2, \cdots, a_{3t}, y$ with $\sum_{i=1}^{3t} a_i = ty$, $\frac{y}{4} < a_i < \frac{y}{2}$. Does there exist a partition $I_1, I_2, \cdots, I_t$ of $\{1, 2, \cdots, 3t\}$ so that $|I_m| = 3$ and $\sum_{i=I_m} a_i = t$, where $m = 1, 2, \cdots, t$?

Given an instance $a_1, a_2, \cdots, a_{3t}, b$ of 3-Partition problem, we construct an instance of the decision version of the maximum weighted tardiness as follows:

$$n_0 = 2t, n_N = 3t$$
$$p_j = 1, d_j = t^2y + t, w_j = ty + 1 j = 1, 2, \cdots, t$$
$$p_j = ty, d_j = t^2y + t, w_j = ty + 1 j = t+1, t+2, \cdots, 2t$$
$$p_j = ta_{j-2t}, d_j = t^2y + t - 1, w_j = \frac{ty(ty+1)}{y-1} j = 2t+1, 2t+2, \cdots, 5t$$
$$k = t^3y + \frac{t(t+1)}{2}, C_1 = t^2y(ty+1)$$

Note that weights of the original jobs are the same. So we assume that $\pi* = (t+1, 1, t+2, 2, \cdots, 2t, t)$ is an optimal job sequence of the original jobs and idle time is not permitted.

The decision asks whether there exists a feasible schedule σ of the above instance of problem $1|\sum \Delta_j(\pi) \leq k|\max w_j T_j$ so that $\max w_j T_j \leq C_1$. The above construct can be done in polynomial time under the unary encoding. We will show that instance of 3-partition problem has a solution iff the instance of the proposed problem has a feasible solution so that $\max w_j T_j \leq C_1$.

$\Rightarrow$ Assume that the instance of 3-partition problem has a solution. Then we will reindex of $a_1, a_2, \cdots, a_{3t}$ so that $\sum_{j=0}^{2} a_{3i-j} = t$, $i = 1, 2, \cdots, t$. Define a tight sequence σ of the scheduling instance, which the jobs will be scheduled as follows:

$$J_{2t+1} \prec J_{2t+2} \prec J_{2t+3} \prec J_1 \prec J_{2t+4} \prec J_{2t+5} \prec J_{2t+6} \prec J_2$$
$$\prec \cdots \prec J_{5t-2} \prec J_{5t-1} \prec J_{5t} \prec J_2 \prec J_{t+1} \prec \cdots \prec J_{2t}.$$

Note that $C_j(\sigma) = C_j(\pi*)$, $j = 1, 2, \cdots, t$. Then

$$\sum_{J_j \in J_0} \Delta_j(\pi*, \sigma) = [t(ty+1)+ty-ty]+[t(ty+1)+2ty-(2ty+1)]+\cdots+[t(ty+1)+t^2y-(t^2y)+t-1] = k,$$

and $\max w_j T_j = (ty+1)[t(2ty+1) - (t^2y+t)] = (ty+1)t^2y$. Hence, σ is a feasible sequence so that $\max w_j T_j \leq C$.

$\Leftarrow$ Assume that problem $1|\sum \Delta_j(\pi) \leq k|\max w_j T_j$ has a feasible schedule σ so that $\max w_j T_j \leq C_1$. Note that $T_j \leq \frac{C}{w_j} = ty - t$ for $J_j \in J_N$. If one of jobs $(J_{t+1}, \cdots, J_{2t})$ is scheduled before the last job of job set J_N, then $T_j = ty - t + 1 \geq ty - t$ for $J_j \in J_N$. So jobs $J_{t+1}, \cdots, J_{2t}$ must be scheduled after job set J_N.

If the h jobs of jobs $J_1, \cdots, J_t$ are scheduled after the first job of jobs $J_{t+1}, \cdots, J_{2t}$ in sequence σ, then the completion time of the job of the h jobs is not more than $t^2y+ty+t-h+1$. Note that the completion time of the job of the h jobs is not more than $t(ty+1)$, the completion times of jobs $J_{t+1}, \cdots, J_{2t}$ in sequence $\pi*$ are $ty, 2ty+1, \cdots, t^2y+t+1$, respectively. In sequence σ, jobs $J_{t+1}, \cdots, J_{2t}$ are scheduled after job sets J_N and the $t-h$ jobs of jobs $J_1, \cdots, J_t$, their completion times are $t^2y+t-h+ty, t^2y+t-h+2ty, \cdots, t^2y+t-h+t^2y$, respectively. Then

$$\begin{aligned}
\sum_{J_j \in J_0} \Delta_j(\pi*, \sigma) &\geq [t^2y + (t-h) + ty - ty] + [t^2y + (t-h) + 2ty - (2ty+1)] \\
&\quad + \cdots + [t^2y + (t-h) + t^2y - (t^2y + t - 1)] + h[t^2y + (t-h) + ty + 1 - (t^2y + t)] \\
&= t^3y + t(t-h) - \frac{t(t-1)}{2} + h(ty - h + 1) \\
&= k + h(ty - h - t + 1)
\end{aligned}$$

Based on definition of 3-Partition problem, we have $y \geq 3$. Note that σ is a feasible sequence, we can obtain $h = 0$. Moreover, $\sum_{j=2t+1}^{2t} \Delta_j(\pi*, \sigma) = k$, $\Delta_j(\pi*, \sigma) = 0$ $(j = 1, 2, \cdots, t)$. That is, jobs $J_1, \cdots, J_t$ occupy the same time intervals in sequences $\pi*$ and σ. Jobs $J_{t+1}, \cdots, J_{2t}$ are scheduled without idle time after job J_t. Note that jobs $J_{t+1}, \cdots, J_{2t}$ have been showed that they are scheduled after job $J_j (\in J_N)$ and $\max w_j T_j \leq C_1$. Then the jobs in J_N can be only processed in the t time intervals $[0, ty], [ty+1, 2ty+1], \cdots, [(t-1)ty + (t-1), t^2y + t - 1]$. This gives a solution of the instance of 3-Partition problem. The result follows. □

5 Relaxing time disruption constraint

This section will present its computation complexity by reducing 3-Partition problem under differen time disruption problem. We will also give an approximation algorithm by 2-Partition problem.

Theorem 3. Problem$1|\Delta_j(\pi) \leq k_j| \max w_j T_j$ is strongly NP-hard.

Proof: We prove problem $1|\Delta_j(\pi) \leq k_j| \max w_j T_j$ by reducing the strongly NP-complete 3-Partition problem [5].

Given an instance $a_1, a_2, \cdots, a_{3t}, b$ of 3-Partition problem, we construct an instance of the decision version of the maximum weighted tardiness as follows:

$$n_0 = 2t, n_N = 3t$$

$$p_j = y, d_j = t^2y + ty, w_j = ty + 1 j = 1, 2, \cdots, 2t$$

$$k_j = +\infty, j = 1, 3, 5, \cdots, 2t-1; k_j = 0, j = 2, 4, 6, \cdots, 2t$$

$$p_j = a_{2t+j}, d_{2t+j} = t^2y + ty - y, w_{2t+j} = ty + 1 j = 1, 2, \cdots, 3t$$

$$C_2 = ty(ty+1)$$

Note that weights of the original jobs are the same. So we assume that $\pi* = (1, 2, \cdots, 2t)$ is an optimal job sequence of the original jobs and idle time is not permitted.

The decision asks whether there exists a feasible schedule σ of the above instance of problem $1|\Delta_j(\pi) \leq k_j|\max w_jT_j$ so that $\max w_jT_j \leq C_2$. The above construct can be done in polynomial time under the unary encoding. We will show that instance of 3-partition problem has a solution iff the instance of the proposed problem has a feasible solution so that $\max w_jT_j \leq C_2$.

$\Rightarrow$ Assume that the instance of 3-partition problem has a solution. Then we will reindex of $a_1, a_2, \cdots, a_{3t}$ so that $\sum_{j=0}^{2} a_{3i-j} = t$, $i = 1, 2, \cdots, t$. We also define a tight sequence σ of the scheduling instance, which the jobs will be scheduled as follows:

$$J_{2t+1} \prec J_{2t+2} \prec J_{2t+3} \prec J_2 \prec J_{2t+4} \prec J_{2t+5} \prec J_{2t+6} \prec J_4 \prec \cdots \prec J_{5t-2} \prec J_{5t-1} \prec J_{5t} \prec J_{2t} \prec J_1 \prec J_3 \prec \cdots \prec J_{2t-1}.$$

Note that $C_j(\sigma) = C_j(\pi*)$, $j = 2, 4, \cdots, 2t$. Then

$$\Delta_j(\pi*, \sigma) \leq k_j, j = 1, 3, \cdots, 2t-1$$

and $\max w_jT_j = ty(ty+1)$. Hence, σ is a feasible sequence so that $\max w_jT_j \leq C_2$.

$\Leftarrow$ Assume that problem $1|\Delta_j(\pi) \leq k_j|\max w_jT_j$ has a feasible schedule σ so that $\max w_jT_j \leq C_2$. Note that $k_j = 0$ for $j = 2, 4, \cdots, 2t$, then $C_j(\sigma) = C_j(\pi*)$. Next, we will show that jobs $(J_1, J_3, \cdots, J_{2t-1})$ is scheduled after the last job of job set J_{2t}. Note that $w_jT_j \leq ty(ty+1)$ for $\forall J_j \in J_N$ then $T_j \leq ty$.

If there exists some job of jobs $J_1, J_3, \cdots, J_{2t-1}$ are scheduled before the last job of job set J_N in sequence σ, then $\max w_jT_j = ty(ty+1) > ty$, for $J_j \in J_N$. So jobs $J_1, J_3, \cdots, J_{2t-1}$ only are scheduled after the last job of job set J_N in sequence σ. Similarly, the new job set J_N must be processed before job J_{2t}. Otherwise, there exists job $J_j \in J_N$, processed after job J_{2t}, then we have $w_jT_j = 2y(ty+1) > ty$. Then the jobs in J_N can be only processed in the t time intervals $[0, y], [2y, 3y], \cdots, [(2t-2)y, (2t-1)y]$. Note that $\sum_{j=2t+1}^{5t} p_j = ty$, which length of each time interval is only y.

Based on definition of 3-Partition problem, we have $y \geq 3$. Note that σ is a feasible sequence, and $\frac{y}{4} < a_i < \frac{y}{2}$. Moreover, $\sum_{j=2t+1}^{2t} \Delta_j(\pi*, \sigma) = k$, $\Delta_j(\pi*, \sigma) = 0$ $(j = 1, 2, \cdots, t)$. Moreover, each time interval only can process three jobs of J_N. This gives a solution of the instance of 3-Partition problem. The result follows. □

Next, we will present a heuristic algorithm of the $1|\Delta_j(\pi) \leq k_j|\max w_jT_j$ problem. Note that constraint condition is time disruption of each job to minimize maximum weighted tardiness. Firstly, we will schedule jobs in J_0 by Algorithm 2. Then jobs in J_0 and J_N are processed by Algorithm 2. Finally, we adjust the original jobs which do not satisfy time disruption k_j.

Algorithm 2

Step 1 Jobs in set J_0 are scheduled by Algorithm 1;

Step 2 Jobs in set J_0 and J_N are scheduled by Algorithm 1;

Step 3 Find the original jobs do not satisfy constraint condition in the sequence obtained by Step 2;

Step 4 Compare the processing times with the original jobs violated constraint condition and the new jobs scheduled before these jobs. If there exists the smaller gap between processing

times of two jobs, then we will interchange their positions, until they satisfy constraint condition of time disruption.

We will firstly present simple numerical experiment to manifest validity of Algorithm 2. Let the original job set be $J_0 = \{J_1, J_2, J_3\}$, the new job set be $J_N = \{J_4, J_5\}$. The processing time, due date, weight and time disruption of each job is given as follows:

$$p_1 = 5, d_1 = 4, w_1 = 1, \triangle_1(\pi*) \leq 10;$$

$$p_2 = 7, d_2 = 4, w_2 = 2, \triangle_2(\pi*) \leq 10;$$

$$p_3 = 10, d_3 = 9, w_3 = 3, \triangle_3(\pi*) \leq 10;$$

$$p_4 = 6, d_4 = 7, w_4 = 2;$$

$$p_5 = 8, d_5 = 2, w_5 = 5.$$

We can obtain optimal sequence by performing Step 1 of Algorithm 1 as follows: $J_2 \prec J_3 \prec J_1$. We can obtain optimal sequence by performing Step 2 of Algorithm 2 as follows: $J_5 \prec J_3 \prec J_2 \prec J_4 \prec J_1$. However, the time disruption of job J_1 and J_2 is more than 10. We obtain a better schedule by interchange jobs, i.e., $J_2 \prec J_3 \prec J_5 \prec J_1 \prec J_4$.

Theorem 3 manifests that there does not exist polynomial time algorithm for problem $1|\Delta_j(\pi*) \leq k_j|\max w_j T_j$. So we will consider inapproximability of the proposed problem. Assume that sequence $\sigma*$ is optimal sequence of problem $1|\Delta_j(\pi*) \leq k_j|\max w_j T_j$. Of course, we only can address their optimal properties and donot pursue to find their optimal sequence.

Theorem 4 Let σ be a job sequence obtained by a polynomial time approximation algorithm of problem $1|\Delta_j(\pi*) \leq k_j|\max w_j T_j$. If $\max w_j T_j(\sigma) \leq \max w_j T_j(\sigma*)$ is satisfied for any given feasible instance, then there is a $\rho(\geq 2)$ approximation algorithm, unless $P = NP$.

Proof: We prove problem $1|\Delta_j(\pi) \leq k_j|\max w_j T_j$ by reducing the NP-complete 2-Partition problem [5].

Partition problem: Given $t+1$ positive integers $a_1, a_2, \cdots, a_t, y$ with $\frac{1}{2}\sum_{i=1}^{t} a_i = y$. Does there exist a partition I_1, I_2 of $\{1, 2, \cdots, t\}$ so that $\sum_{i=I_1} a_i = \sum_{i=I_2} a_i = y$?

Given an instance $a_1, a_2, \cdots, a_t, y$ of 2-Partition problem, we construct an instance of the decision version of the maximum weighted tardiness as follows:

$$n_0 = 4, J_0 = \{J_1, J_2, J_3, J_4\}; n_N = t, J_N = \{J_5, J_6, \cdots, J_{t+4}\}$$

$$p_1 = y, d_1 = 0, k_1 = +\infty, w_1 = 1$$

$$p_2 = y, d_2 = 0, k_2 = 0, w_2 = 1$$

$$p_3 = y, d_3 = 0, k_1 = +\infty, w_3 = 1$$

$$p_4 = vy, d_4 = 0, k_4 = 0, w_4 = 1$$

$$p_j = a_{j-4}, d_j = -vb, w_j = 1 j = 5, 6, \cdots, t+4$$

Assume that the original optimal sequence $\pi* = \{J_1, J_2, J_3, J_4\}$, and v is a sufficiently large positive number.

Next, we will show that partition problem can be solved in polynomial time, if and only if $\rho \leq 2$. This means that there is a $\rho(\geq 2)$ approximation algorithm for problem $1|\Delta_j(\pi*) \leq$

$k_j|\max w_jT_j$, unless unless $P = NP$. Since $k_2 = k_4 = 0$, job J_2 and job J_4 must be processed in the same time intervals with sequence σ and sequence $\pi*$, i.e., they will be scheduled in the time intervals $[y, 2y], [3y, 3y + vy]$. So jobs in J_N must be processed in time intervals $[0, y], [2y, 3y]$ or from $3y + vy$.

The decision asks whether there exists a feasible schedule σ of the above instance of the problem $1|\Delta_j(\pi) \leq k_j|\max w_jT_j$. The above construct can be done in polynomial time under the unary encoding. We will show that instance of partition problem has a solution iff the instance of the proposed problem has a feasible solution.

Assume that the instance of partition problem has a solution, and (I_1, I_2) is a partition of $1, 2, \cdots, t$, i.e., $\sum_{i=I_1} a_i = \sum_{i=I_2} a_i = y$. The following four cases will be discussed:

- $\forall i \in I_i$, job J_{i+4} is processed in time interval $[0, y]$; $\forall i \in I_i$, job J_{i+4} is processed in time interval $[0, y]$. Job J_1 and job J_3 are processed from time $3y + vy$. So we have $\max w_jT_j = 5y + vy$.

- $\forall i \in I_i$, job J_{i+4} is processed in time interval $[0, y]$; $\forall i \in I_i$, job J_{i+4} is processed from time $3y + vy$. Job J_1 are processed in time interval $[4y + vy, 5y + vy]$. So we have $\max w_jT_j = 5y + vy$.

- $\forall i \in I_i$, job J_{i+4} is processed in time interval $[2y, 3y]$; $\forall i \in I_i$, job J_{i+4} is processed from time $3y + vy$. Job J_3 are processed in time interval $[4y + vy, 5y + vy]$. So we have $\max w_jT_j = 5y + vy$.

- The original jobs J_0 are scheduled in the same position with $\pi*$ and σ. Jobs in J_N are processed from time $3y + vy$. So we have $\max w_jT_j = 5y + vy$.

It is obvious that the instance of the first case is an optimal sequence, say $\sigma*$, and instances of the second case, the third case and the fourth case are feasible sequences generating by polynomial time approximation algorithm A, say σ. Then we have $\max w_jT_j \geq 2vy$, and

$$lim_{v\to+\infty}\frac{w_jT_j(\sigma)}{w_jT_j(\sigma*)} > lim_{v\to+\infty}\frac{2vy}{vy} = 2.$$

If $\rho < 2$, then jobs in J_N will be processed in time intervals $[0, y]$ and $[2y, 3y]$ by the corresponding polynomial time approximation algorithm. So partition problem has a solution if and only if $1|\Delta_j(\pi) \leq k_j|\max w_jT_j$ has also a solution by reducing in polynomial time, unless $P = NP$. So the result follows. □

Algorithm 2'

Step 1 Jobs in set J_0 are scheduled by Algorithm 1, say $\sigma = \{J_1, J_2, \cdots, J_{n_0}\}$;

Step 2 Jobs in set J_N are scheduled by Algorithm 1, say $\sigma_1 = \{J_{n_0+1}, J_{n_0+2}, \cdots, J_{n_0+n_N}\}$;

Step 3 Temporary insert job J_{n_0+r} from σ_1, into σ according Algorithm 1 and update σ. If the inserting operation does not violate $\triangle_j(\pi*) \leq k_j$, update σ and $\sigma_1 = \sigma_1\backslash\{J_{n_0+r}\}$. Otherwise we keep the sequence σ and append job J_{n_0+r} at the end of sequence σ. Let $r := r+1$, if $r \leq n_N$, then turn to Step 3. Otherwise, go to Step 4.

Step 4 Output the final schedule σ as the solution.

6 Numerical Experimentation

Several computational experiments are conducted to determine the performance of the algorithm two (Ag2) and its subsequent two-stage improvement. The main reason is that the measurement criterion is very sensitive and easily affected by the factor of due dates. Therefore, in order to get a good quality of solutions, the final solutions of proposed Ag2 will be improved by three local search methods such as the pairwise interchange, backward-shifted re-insertion, and forward-shifted re-insertion (refer to [3]) for the first stage. They are recorded as Ag2p, Ag2b, and Ag2f, respectively. For the second stage, we reset Ag2p, Ag2b, and Ag2f as three initials in the simulation annealing (SA) to refine them. They are recorded as $Ag2p_SA$, $Ag2b_SA$, and $Ag2f_SA$, respectively. After our tuning parameters tested, the initial temperature, final temperature, the number of improvements and cooling factor in SA are set as $(0.15, 0.00001, 20, 0.94)$ in the following instances tested. All proposed algorithms are coded in FORTRAN.

To show the impact of parameters such as the number of n_v, due dates, and job size, we design some tested scheme as follows : the processing time and its weight of job J_j are respectively generated from the uniform distribution $U(1, 10)$, while the disruption time for each new job is created from another uniform distribution $U(0, v_1 \sum_{j \in n_N} p_j)$ where v_1 was a control variable. The value of v_1 is set at $0.4, 0.5$, and 0.6. In addition, the due dates are generated from $TP * U(1 - \tau - R/2, 1 - \tau + R/2)$, where $TP = \sum_{j=1}^{n} p_j$, and τ and R are denoted as the tardiness factor and the range of the due dates (refer to [4]). The values of (τ, R) are set at $(0.5, 0.75), (0.5, 0.5), (0.5, 0.25), (0.25, 0.25), (0.25, 0.5)$, and $(0.25, 0.75)$. One hundred instances are generated for each case. In total of 2400 instances (i.e., $n * \tau * R * 100 = 4 * 2 * 3 * 100 = 2400$) are tested.

For the proposed algorithms based on small numbers of jobs ($n == 10$ and 11, where $10 = 8 + 2$ or $7 + 3$; $11 = 9 + 2$ or $8 + 3$), we reported the mean and maximum error percentages in which the average error percentage (AEP) was defined as $AEP = \frac{(H_i - O*)}{O*} * 100\%$, where $H*$ was the value of the maximum weighted tardiness yielded from Ag2, Ag2p, Ag2b, Ag2f, $Ag2p_SA$, $Ag2b_SA$, and $Ag2f_SA$ and $O*$ was the value of the maximum weighted tardiness yielded from the all possible permutation method. Meanwhile, for the proposed algorithms based on large numbers of jobs ($n = 100$ and 120, where $100 = 80 + 20$ or $70 + 30$; $120 = 100 + 20$ or $90 + 30$), we recorded the mean and maximum relative percentages of deviation (RPDs) in which the RPD was defined as $RPD = [\frac{(H_i - H*)}{H*}] * 100\%$, where H_i was the value of the maximum weighted tardiness yielded from seven algorithms and $H*$ was the smallest objective value selected from among seven algorithms.

As shown in Table 1, it can be seen that in the same job size level, the AEP of Ag2 declines sharply after it is improved by three local search improvements, where Ag2p performs best than two others. It can also be observed that the average AEP of Ag2b is high up to 10.16. However, the average AEP of Ag2b can further decrease to 0.07 after they are improved by SA. Regarding the impact of τ, on an average, all the AEPs of Ag2, Ag2p, Ag2b and Ag2f on a small value of τ are smaller than they are on a big value of τ, while the AEPs of Ag2, Ag2b and Ag2f increases as the value of R increases. Only the AEP of Ag2 increases as v_1 increases. For the impact of n_N, it can be found that the AEPs of Ag2, Ag2p, and Ag2b increase, but the AEP of Ag2f decrease as n_N increases. Overall, Table 1 also reported that on an average, the AEP of Ag2 decreases sharply as n_N increases by one only, but the AEPs of $Ag2p_SA$, $Ag2b_SA$, and $Ag2f_SA$ (in an average, all of them are less than 0.11%) are very stable and robust no matter that n, τ, R, v_1, or n_N changes.

Regarding the performances of all proposed seven algorithms on the big number of jobs, it can be observed in Table 2 that on an average, all the RPDs of seven algorithms at a small value of τ are bigger than they are at a big value of τ; but for the n_N, the overall situation is just the opposite. The RPDs of seven algorithms increase as the value of v1 increases. Regarding the impact of R, it can be seen that the RPDs of Ag2f, Ag2b, $Ag2f_SA$ and $Ag2b_SA$ increase as R increases. Furthermore, Table 2 also reported that on an average, except $Ag2f_SA$, the RPDs of proposed six algorithms decreases, as n increases. Overall, as shown in Fig 1 and Fig 2, $Ag2p_SA$ did very well among seven algorithms in term of the AEPs and RPDs.

parameter	value	Ag2	Ag2p	Ag2f	Ag2b	Ag2p_SA	Ag2f_SA	Ag2b_SA
n	10	164.29	1.13	2.17	7.67	0.07	0.05	0.05
	11	93.43	0.80	2.31	10.16	0.07	0.06	0.07
τ	0.25	203.51	1.39	3.35	14.21	0.06	0.05	0.06
	0.5	54.21	0.54	1.13	3.62	0.08	0.05	0.06
R	0.25	72.80	0.50	0.90	3.26	0.10	0.04	0.05
	0.5	134.71	1.54	2.20	7.35	0.05	0.07	0.06
	0.75	179.07	0.84	3.62	16.15	0.06	0.04	0.08
v_1	0.4	111.51	0.62	1.81	10.50	0.05	0.06	0.06
	0.5	133.13	0.54	0.95	7.03	0.05	0.07	0.06
	0.6	141.95	1.73	3.96	9.22	0.11	0.03	0.06
n_N	2	118.34	0.29	2.46	3.41	0.06	0.05	0.05
	3	139.38	1.63	2.02	14.42	0.08	0.06	0.07

Table 1. The performances of AEPs of seven algorithms over different parameters

parameter	value	Ag2	Ag2p	Ag2f	Ag2b	Ag2p_SA	Ag2f_SA	Ag2b_SA
n	100	87.00	6.45	23.93	29.42	1.48	2.83	3.54
	120	71.35	4.67	21.62	25.17	1.05	3.08	3.45
τ	107.74	7.52	29.51	35.94	1.61	3.66	4.44	107.74
	50.60	3.61	16.04	18.64	0.91	2.25	2.55	50.60
R	76.65	4.65	14.90	21.66	1.02	2.01	2.77	76.65
	83.16	6.43	24.71	28.25	1.61	3.21	3.40	83.16
	77.71	5.60	28.71	31.97	1.16	3.64	4.32	77.71
v_1	60.95	3.60	14.29	18.09	0.84	1.90	2.18	60.95
	79.76	5.60	22.61	27.17	1.26	2.91	3.42	79.76
	96.81	7.49	31.42	36.63	1.68	4.06	4.89	96.81
n_N	41.91	3.17	11.37	13.28	0.68	1.01	1.46	41.91
	116.43	7.96	34.18	41.31	1.85	4.90	5.54	116.43

Table 2. The performances of RPDs of seven algorithms over different parameters
Fig 1 The performances of AEPs of seven algorithms
Table 2. The performances of RPDs of seven algorithms over different parameters

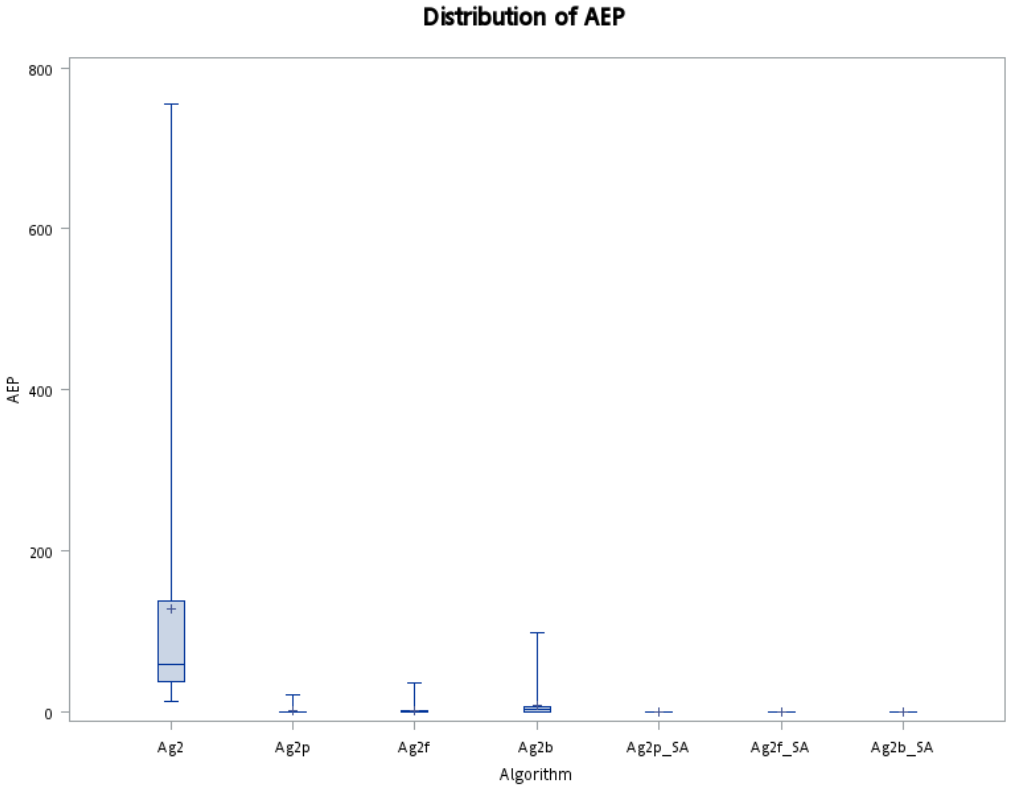
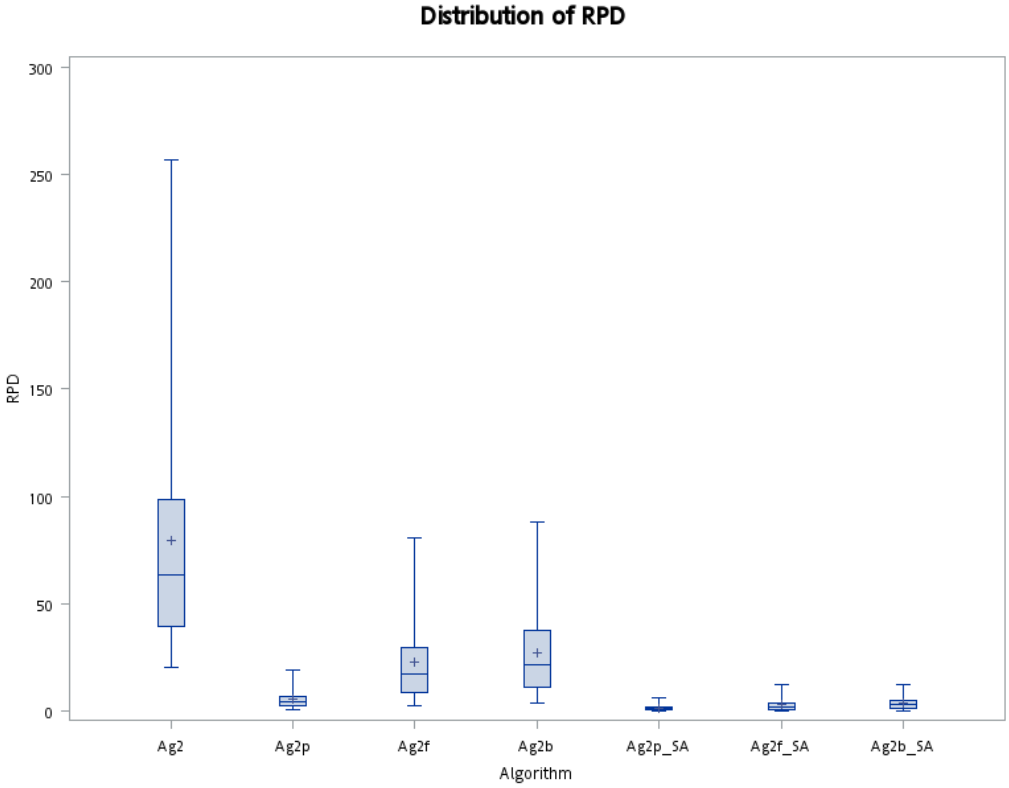


Fig 2 The performances of RPDs of seven algorithms



7 Conclusion

In this paper, two rescheduling problems are studied to allow for the arrival of new jobs, taking into account the effect of time disruptions on a previously planned optimal schedule. Rescheduling means that a set of original jobs has already been scheduled to minimize maximum weighted tardiness, then a new set of jobs arrives and creates a time disruption. Two time disruption problems will be involved: under an upper restriction on the total time disruption and relaxing time disruption restriction. For the former case, We give the proof of strongly NP-hard by reducing 3-Partition problem. For latter case, we present strong NP-hard proof and provide a 2-approximation polynomial time algorithm, and show that, unless $P = NP$, the proposed problem can not have an approximation polynomial time algorithm with a performance ratio less than 2. Finally, we also present some extensive numerical studies to evaluate the performance of these algorithms. Future research may stimulate rescheduling models to mitigate the effects of the disruptions that occur frequently in manufacturing practice. Another future research can be extended to more general machine environment.

Acknowledgements—-This work was supported Project Supported by the National Natural Science Foundation of China (11971443), the Chongqing Municipal Science and Technology Commission of Natural Science Fund Projects (cstc2018jcyjAX0631), and Key Research fund Projects of Chongqing Graduate Education and Teaching Reform (yjg182019).

References

[1] Akram K, Kamal K(2016). Fast simulated annealing hybridized with quenching for solving job shop scheduling problem. Applied Soft Computing, 49: 510-523.

[2] Aytug H, Lawley MA, McKay K, Mohan S, Uzsoy R (2005). Executing production schedules in the face of uncertainties: a review and some future direction. European Journal of Operation Research, 161: 86-110.

[3] Della Croce F, Narayan V, Tadei R(1996). The two-machine total completion time flow shop problem, European Journal of Operational Research, 90: 227-237.

[4] Fisher ML(1976). A dual algorithm for the one-machine scheduling problem, Mathematical Programming, 11: 229-251.

[5] Garey MR and Johnson DS(1979). Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Franciso.

[6] Hall NG and Potts CN(2004). Rescheduling for new orders. Opereations Research, 52: 440-453.

[7] Hao X, Gen M, Lin L, Suer GA(2017). Effective multiobjective EDA for bi-criteria stochastic job-shop scheduling problem. Journal of Intelligent Manufacturing, 28(3), 833-845.

[8] Hoogeveen H, Lente C, T kindt V (2012) Rescheduling for new orders on a single machine with setup times. European Journal of Operation Research, 223: 40-46.

[9] Kang YH and Zhang XG(2018). A Single Machine Rescheduling Problems with Deteriorating Jobs Under Sequence Disruptions. Journal of Systems Science and Complexity, 38(4): 511-520.

[10] Katragjini K, Vallada E, Ruiz R (2013). Flow shop rescheduling under different types of disruption. Internation Journal of Production Research, 51:780-797.

[11] Kirkpatrick S, Gelatt C, Vecchi M (1983). Optimization by simulated annealing. Science, 220/4598:671-680.

[12] Liu L, Zhou H (2015). Single-machine rescheduling with deterioration and learning effects against the maximum sequence disruption. Internation Journal of System Science, 46:2640-2658.

[13] Lawler EL. (1973). Optimal sequencing of a single machine subject to precedence constraints, Management Science, 19: 544-546.

[14] Luo WC, Luo TB, Goebel R, Lin GH(2018). Rescheduling due to machine disruption to minimize the total weighted completion time. Journal of Scheduling, 21:565-578.

[15] Moschakis IA, Karatza HD(2015). Multi-criteria scheduling of Bag-of-Tasks applications on heterogeneous interlinked clouds with simulated annealing. Journal of Systems and Software, 101: 1-14.

[16] Taghi Assadi M, Bagheri M(2016). Differential evolution and Population-based simulated annealing for truck scheduling problem in multiple door cross-docking systems. Computers and Industrial Engineering, 96: 149-161.

[17] Teghem J, Tuyttens D (2014). A bi-objective approach to reschedule new jobs in a one machine model. International Transactions in Operational Research, 21: 871-898

[18] Valledora P, Gomez A, Prioreb P and Puente J(2018). Solving multi-objective rescheduling problems in dynamic permutation flow shop environments with disruptions. International Journal of Production Research, 2018, 56(19): 6363-6377.

[19] Vieira GE, Herrmann JW, Lin E (2003). Rescheduling manufacturing systems: a framework of strategies, policies and methods. Journal of Scheduling, 6: 39-62.

[20] Wu CC, Wang DJ, Cheng, SR, Chung IH, Lin WC (2018). A two-stage three-machine assembly scheduling problem with a position-based learning effect. International Journal of Production Research, 56(9): 3064-3079.

[21] Xiao J, Yang C, Zheng L, Gupta JND(2015). A hybrid Lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times. Computers and Operations Research, 63: 72-82.

[22] Yang J(2018). Rescheduling a Single Machine with Various Objectives. Industrial Engineering & Management Systems, 17(4): 805-818.

[23] Yang B.B.(2009). Single machine rescheduling with new jobs arrivals and processing time compression. The International Journal of Advance Manufacture Technology, 34: 378-384.

[24] Yuan J.J. and Mu Y.D.(2007). Rescheduling with release dates to minimize makespan under a limit on the maximum sequence disruption. European Journal of Operation Research, 182: 936-944.

[25] Zhao QL, Lu LF, Yuan JJ.(2016). Rescheduling with new orders and general maximum allowable time disruptions 4OR-Q Journal of Operation Research, 14 (3):261-280.

[26] Zhao CL, Tang HY (2010). Rescheduling problems with deteriorating jobs under disruptions. Applied Mathematical Modelling, 34:238-243.

[27] Zhao QL, Yuan JJ (2013). Pareto optimization of rescheduling with release dates to minimize makespan and total sequence disruption. Journal of Scheduling, 16:253-260

MISTA 2019

Reentrant Flow Shop with Identical Jobs and Unit Processing Times

Ehud Ikar · Tal Grinshpoun · Hagai Ilani · Elad Shufan

Abstract The article focuses on the problem of a reentrant flow shop with unit processing times and makespan minimisation objective. We correct two results published a few decades ago regarding the problem. Our algorithmic effort is based on two simple polynomial algorithms. We point out the advantages of each of the two algorithmic solutions. In some cases they lead to an optimal solution, but they are generally not optimal. In fact, the complexity of the problem is still unknown. The article is a first step towards understanding a wider real-life problem of identical jobs in a semi-mass production line.

Keywords Reentrant flow shop · Identical jobs · Unit processing times · Semi-mass production

1 Introduction

We tend to divide the world of manufacturing into two main groups: *custom manufacturing* and *mass production*. In custom manufacturing, the quantity of produced items is usually small. In most cases, the production is highly flexible to enable customisation to the specific needs of the clients. As a consequence, custom manufacturing is mostly performed by humans. Scenarios of

Ehud Ikar
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: ehud.ikar@msmail.ariel.ac.il

Tal Grinshpoun
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: talgr@ariel.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Elad Shufan
Physics Department, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: elads@sce.ac.il

mass production, however, are often characterised by high automation, and usually include a fixed order of operations, uniform and known duration of operations, lack of flexibility in the process, and absorption of costs that are derived from defective production units [5]. A representative example of mass production is a mobile phone manufacturing line that can produce millions of items per year.

In addition to the above rather dichotomous division, there is another group called *semi-mass production*, which refers to manufacturing lines that produce from hundreds to thousands of items per year. Such lines usually involve mixture of manual and automatic manufacturing, with some flexibility in the production process. A manufacturing line for aircraft computers is a representative example of semi-mass production.

This article is inspired by a semi-mass production problem. As a first step, we consider production of identical objects, i.e., identical jobs. The manufacturing of each job consists of a set of operations with known precedences between them. Herein we treat sequential precedences, in a manner similar to a flow shop problem, where the operations are performed one by one, in a known order. The production line includes a number of workers, where each worker is qualified to perform several operations. The objective is to find a schedule that minimises the *makespan* C_{max}, i.e., the total completion time of all the jobs.

In general, it is possible that several workers are qualified to perform the same operation. We present this problem, but mostly deal with a special case for which any operation is processed by only one worker. The obtained problem is the known reentrant flow shop problem with identical jobs, denoted $Fm|reentrant, p_{ij} = p_i|C_{max}$. The term *reentrant* indicates that a given machine can be visited by a job more than once. The order by which the jobs visit the machines is called the reentrant pattern. The reentrant flow shop problem was first proposed by Graves [9]. The problem is discussed intensively in chapter 9 of Emmons et. al. [7]. The problem $Fm|reentrant|C_{max}$ is shown to be NP-hard even when there are only two machines ($m = 2$). Several specific reentrant patterns, such as cyclic-reentrant [15,2], chain-reentrant [23, 22], hub-reentrant [12,24] and V-reentrant [13], are discussed in the literature. Some authors refer to the problem as the cycle-shop problem [21], which is cumbersome because it can be confused with a different problem of cyclic (periodic) scheduling [4,10,16].

Here we concentrate in the more specific settings, where all operations have unit processing times, i.e., $Fm|reentrant, p_{ij} = 1|C_{max}$. We consider two polynomial algorithms for the problem. The first algorithm is termed the operation-by-operation (ObO) algorithm, because it gives priority to operations according to their order. The second algorithm, which we call the job-by-job (JbJ) algorithm, prioritises the jobs according their order. The problem notations, along with a full description of these two algorithms, are given in Section 2.

Regarding the complexity of $Fm|reentrant, p_{ij} = 1|C_{max}$, it was argued that the ObO algorithm leads to a minimum makespan in linear time. This

leads to the conclusion that the problem is polynomial [20, 21]. In Section 4 we show an example that contradicts the optimality of the ObO algorithm. As we prove in Section 3 the ObO algorithm is optimal for several specific reentrant patterns. Hence, for some patterns, the problem is indeed polynomially (linearly) solvable. Another polynomially solvable case is of a regular flow shop with identical jobs and preemptions allowed [7], $Fm|prmp, p_{ij} = p_i|C_{max}$. This is in fact a special case of $Fm|reentrant, p_{ij} = 1|C_{max}$, where operations that are processed by one specific machine are consecutive operations. The optimal schedule is obtained by permutation scheduling in any order and with no need for preemptions. Many versions of the reentrant flow shop problem were considered and classified by their complexity [14, 21], including various objective functions and characteristics. Some of these versions are NP-hard even when unit processing times are considered. In general, the complexity of $Fm|reentrant, p_{ij} = 1|C_{max}$ remains open.

One of the patterns that the ObO algorithm does solve optimally is the V-reentrant pattern. It was claimed that the JbJ algorithm leads to an optimal schedule, for two objectives – C_{max} and $\sum C_j$ [13]. In Section 3 we correct that result and show that while the ObO algorithm is optimal for the C_{max} objective, the JbJ algorithm fails to generally return an optimal schedule. Nevertheless, as shown in Section 4, the JbJ algorithm has an advantage when scheduling problems with general reentrant patterns and a large number of jobs. We raise a conjecture which, if true, leads to the conclusion that the ratio between the JbJ makespan and the optimal makespan tends to unity as the number of jobs increases. Periodicity characteristics of the JbJ schedule are also discussed in Section 4. If integral preemption is allowed than these results are also relevant for problems with identical jobs, but not necessarily unit-time. This is discussed in Section 5.

2 Model and Basic Algorithms

2.1 Definitions

We consider a system that includes $J = \{1, 2, \ldots, j, \ldots, n\}$ identical jobs (product units). In order to produce one job it is necessary to perform a series of operations $O = (1, 2, \ldots, i, \ldots, l)$. The operations are performed by $M = \{1, 2, \ldots, k, \ldots, m\}$ machines (workers), according to their order in O. Each machine k has a set of operations it can perform, denoted by O^k. The i-th operation of job j is denoted by O_{ij}. The set of all i-th operations of all the jobs is denoted by O_i. The processing time of operation O_{ij} is p_{ij}. For identical jobs, p_{ij} is job-independent, i.e., $p_{ij} = p_i$. In this article we regard a further simplification, taking $p_i = 1$. Consider next an example.

Example 1 $[J = \{1, 2, 3\}, O = (1, 2, 3, 4, 5), M = \{1, 2\}, O^1 = \{1, 2, 3, 4\}, O^2 = \{4, 5\}]$. Three product units should be produced; five sequential operations are required in order to produce a single unit. The system has two workers, where the first worker is qualified to perform the first four operations, and the

second worker is able to perform the last two operations. As evident, the fourth operation may be performed by either of the workers. Because the operations are sequential and the jobs identical this problem can be also described as follows: $[n = 3 \text{ jobs}, m = 2 \text{ machines}, O^1 = \{1, 2, 3, 4\}, O^2 = \{4, 5\}]$.

The above model is a generalisation of the reentrant flow shop, in which each operation can be performed by exactly one machine (worker). More formally, in a reentrant flow shop for each pair of machines $a, b \in M$, it follows that $O^a \cap O^b = \emptyset$. The system of Example 1 is clearly not a reentrant flow shop, because of the fourth operation ($O^1 \cap O^2 = \{4\} \neq \emptyset$). Nonetheless, a reentrant flow shop is clearly a special case of the above defined system. For instance, by modifying the sets of operations that can be performed by each worker, the problem of Example 1 can be turned into a reentrant flow shop problem.

Example 2 $[n = 3, m = 2, O^1 = \{1, 3, 5\}, O^2 = \{2, 4\}]$. In this system each operation can be performed by exactly one worker, and is thus a reentrant flow shop problem.

Reentrant problems are usually presented according to the order of the jobs in the different machines, which is called the reentrant *pattern* and is denoted by ϕ (see [7]). By this notation Example 2 can be presented by $[n = 3, m = 2, \phi = (1, 2, 1, 2, 1)]$. Furthermore, the number of machines can be deduced from the pattern, and can also be omitted. This is a more compact representation for reentrant flow shop problems. However, it cannot describe the more general case, e.g., the case presented in Example 1. Hereinafter, we consider a reentrant flow shop and hence use the ϕ notation.

2.2 Algorithms

We will use two different greedy algorithms to solve reentrant problems: the JbJ algorithm and the ObO algorithm, which are described next. The JbJ algorithm greedily processes one job after the other, i.e., it processes all operations of a job, as early as possible, before continuing to the next job. The resulting schedule is called the JbJ schedule.

Alternatively, the ObO algorithm processes one operation after the other, i.e., it processes a particular operation on all jobs, as early as possible, before continuing to the next operation. The ObO algorithm maintains the same order of jobs for every operation processed. The resulting schedule is called the ObO schedule, also termed an operation-block schedule by Timkovsky [21].

Consider the scheduling problem of Example 2. Figure 1 depicts the first, second and last processing stages according to the JbJ algorithm, leading to $C_{max} = 11$.

Figure 2 shows the the first, second and last processing stages according to the ObO algorithm, skipping stages three and four. The obtained makespan is $C_{max} = 9$. It is evident that in this example the makespan of the ObO

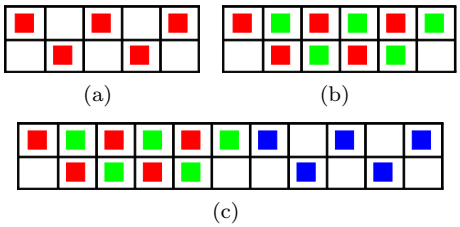


Fig. 1 Processing of Example 2 according to the JbJ algorithm after (a) one job, (b) two jobs, and (c) three jobs, i.e., the final JbJ schedule. Different jobs have different colours.

schedule is shorter than that of the JbJ schedule (9 vs. 11). In fact, the ObO schedule is clearly optimal for this example, because it involves continuous work of machine 1 from the beginning without any idle times.

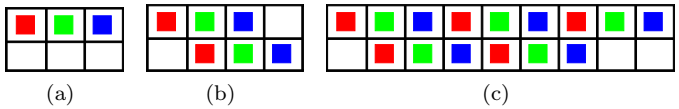


Fig. 2 Processing of Example 2 according to the ObO algorithm after (a) one operation, (b) two operations, and (c) five operations, i.e., the final ObO schedule.

3 Special Reentrant Patterns

This section deals with special patterns of the reentrant flow shop. All these patterns are presented in the Emmons and Vairaktarakis book about flow shop scheduling [7], which devotes one of its chapters to reentrant flow shops. We show that for unit processing times with the objective of makespan minimisation, the ObO algorithm is optimal for all the presented patterns.

The patterns studied in this section are schematically shown in Figure 3 and include:

1. V-reentrant, $\phi = (1, 2, \ldots, m-1, m, m-1, \ldots, 2, 1)$
2. Cyclic-reentrant, $\phi = (1, 2, \ldots, m, 1, 2, \ldots, m, \ldots, 1, 2, \ldots, m)$
3. Chain-reentrant, $\phi = (1, 2, \ldots, m, 1)$
4. Hub-reentrant, $\phi = (1, 2, 1, 3, \ldots, 1, m, 1)$

3.1 The V-reentrant Pattern

The V-reentrant pattern $\phi = (1, 2, \ldots, m-1, m, m-1, \ldots, 2, 1)$, introduced first by Lev and Adiri [13], consists of an operation m that is preceded by a sequence of preliminary tasks, called *forward* operations, and followed by the same tasks in reverse order, i.e., *backward* operations.

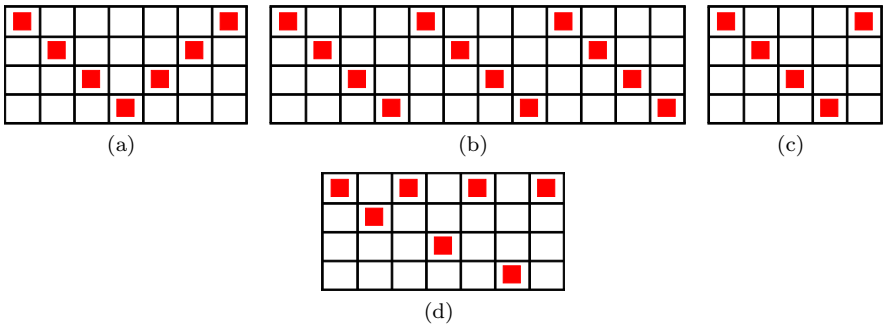


Fig. 3 Special patterns of a reentrant flow shop: (a) V-reentrant, (b) cyclic-reentrant with three cycles, (c) chain-reentrant, and (d) hub-reentrant.

An example of such pattern is the replacement of a damaged internal part [13]. Parts blocking access need to be removed first, then the damaged part is replaced, and finally the removed parts need to be restored in reverse order.

Another example, taken from the world of systems engineering, is the *Vee*-chart [8] that describes hierarchical stages of a project. For instance, $System \rightarrow Subsystem \rightarrow Component \rightarrow Segment \rightarrow Item$. The forward operations consist of the stages' design, starting from the system-level design. After completing the design, the integration and testing phase begins in reverse order. First, the low levels are tested, and only in the final stage integration and tests are performed at the system level.

Denote by $Fm|V\text{-}reentrant, p_{i,j} = 1|C_{max}$ a V-reentrant flow shop with m machines, unit processing times, and makespan minimisation objective.

Theorem 1 *The ObO algorithm is optimal for* $Fm|V\text{-}reentrant, p_{i,j} = 1|C_{max}$. *The makespan is given by* $C_{max} = 2(n + m - 2)$, *for* $n > 1$.

Proof We first show that an ObO schedule leads to $C_{max} = 2n + 2(m - 2)$, for $n > 1$. The argument is simple, and may be more clear to the reader by looking at Figure 4, which shows the ObO schedule for an example V-reentrant flow shop problem. In the ObO algorithm, the set of operations O_1 is scheduled first and is completed at time n. The next sets of forward operations $O_2, O_3, \ldots, O_{m-1}, O_m$ that are processed on machines $k = 2, 3, \ldots, m-1, m$, are completed at time $n + 1, n + 2, \ldots, n + m - 2, n + m - 1$, respectively. The first completed set of backward operations is O_{m+1}, which is processed on machine $m - 1$. It starts at time $n + m - 2$ (here $n > 1$) and is completed at time $t := 2n + m - 2$. The other sets of backward operations $O_{m+2}, O_{m+3}, \ldots, O_{2m-1}$ on machines $k = m - 2, m - 3, \ldots, 1$ are completed at times $t+1, t+2, \ldots, t+(m-2)$, respectively. Thus, $C_{max} = t + (m - 2) = 2n + 2(m - 2)$.

We next show that the above obtained C_{max} value is optimal. Given any schedule, let Δ_k be the time difference between the completion time of the

last operation on machine k and the starting time of the first operation on the same machine. Clearly, the V-reentrant pattern yields $C_{max} = \Delta_1$. In addition, according to the V-reentrant pattern, Δ_{k-1} is larger than Δ_k by at least two time units (for any schedule). Hence, $\Delta_1 \geq \Delta_2 + 2 \geq \Delta_3 + 4 \geq \ldots \geq \Delta_{m-1} + 2(m-2)$. Moreover, because each machine $k = 1, 2, \ldots, (m-1)$ processes $2n$ operations, it follows that $\Delta_k \geq 2n$ for any $k \neq m$. In particular, $\Delta_{m-1} \geq 2n$. Therefore, $C_{max} = \Delta_1 \geq \Delta_{m-1} + 2(m-2) \geq 2n + 2(m-2)$. Because the makespan of the ObO schedule is $C_{max} = 2n + 2(m-2)$ (for $n > 1$), it follows that the ObO algorithm is optimal. ■

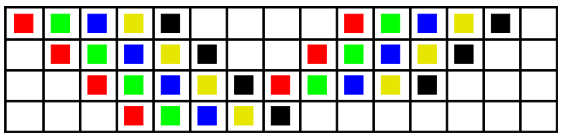


Fig. 4 An ObO schedule for a unit-time V-reentrant flow shop with four machines and five jobs.

In Figure 4 we show an example with $m = 4$ machines and $n = 5$ jobs. The four sets of forward operations are completed at times $5, 6, 7, 8$ and the remaining three sets of backward operations are completed at times $t = 12$, $t + 1 = 13$, $t + 2 = 14$, i.e., $C_{max} = 14$. The JbJ schedule, presented in Figure 5, leads to a makespan of 15. The JbJ algorithm was mistakenly reported to be optimal for $Fm|V\text{-}reentrant, p_{i,j} = 1|C_{max}$ [13], but it is so only for even n. For odd n, the JbJ makespan is actually larger than the optimal makespan by one time unit.

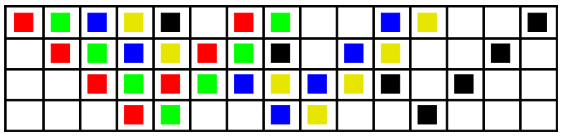


Fig. 5 A JbJ schedule for a unit-time V-reentrant flow shop with four machines and five jobs.

3.2 The Cyclic-reentrant Pattern

The cyclic-reentrant pattern $\phi = (1, 2, \ldots, m, 1, 2, \ldots, m, \ldots, 1, 2, \ldots, m)$ consists of making r passes, where r is the number of cycles, through all machines in order.

An example of such pattern is the assembly of electronic circuits stacked on top of each other [7]. Every time a new circuit is attached, the same set of machines is revisited.

One may also think of a cycle as the production of a single product within a *batch* [1], where the number of cycles r corresponds to the number of products in each batch.

Denote by $Fm|r\text{-}cyclic\text{-}reentrant, p_{i,j} = 1|C_{max}$ a cyclic-reentrant flow shop with m machines, r cycles, unit processing times, and makespan minimisation objective.

Theorem 2 *The ObO algorithm is optimal for* $Fm|r\text{-}cyclic\text{-}reentrant, p_{i,j} = 1|C_{max}$. *The makespan is given by* $C_{max} = \begin{cases} nr + m - 1 & \text{if } n \geq m \\ mr + n - 1 & \text{otherwise} \end{cases}$.

Proof **Case 1:** $n \geq m$. When scheduled according to ObO, each machine starts as early as possible and works continuously (Figure 6(a)). This is because for each cycle, the last operation of the first machine ends after (or, for $n = m$, at the same time as) the first operation of the last machine. The obtained schedule is therefore optimal. Machine $k = 1$ works for r rounds, each round of size n, and therefore finishes working at time nr. Machines $k = 2, 3, \ldots, m$ finish working at times $nr + 1, nr + 2, \ldots, nr + (m - 1)$, respectively. Hence, $C_{max} = nr + m - 1$.

Case 2: $n < m$. In each of the r cycles, n operations are processed by each of the m machines. Denote by $O_{j,k,c}$ an operation of job $j = 1, 2, \ldots, n$ processed by machine $k = 1, 2, \ldots, m$, in cycle $c = 1, 2, \ldots, r$. Consider machine $k = 1$ of an ObO schedule. In the first cycle it finishes working at time n (with operation $O_{n,1,1}$). But, the earliest time that the second round can start is not n but rather m (because operation $O_{1,1,2}$ can start only after operation $O_{1,m,1}$, which finishes at time m). The machine therefore works intermittently, with operations $O_{1,1,c}$, $c = 1, 2, 3, \ldots, r$, starting at times $0, m, 2m, \ldots, (r - 1)m$ (Figure 6(b)). The same is true for the next machines, with time shifts of one unit when comparing consecutive machines. Therefore, the last machine begins its last round ($c = r$) at time $(r - 1)m + (m - 1)$ and finishes working (with the last operation of the total schedule) after n more time units. Hence, $C_{max} = (r - 1)m + (m - 1) + n = mr + n - 1$.

We next show that C_{max} cannot be smaller, i.e., ObO is optimal also from $n < m$. For any schedule, the completion time of operation $O_{j,k,c+1}$ is larger than that of operation $O_{j,k,c}$ by at least m time units. Therefore, the completion time of the first job that finishes its processing on machine 1 is not smaller than $t := m(r - 1) + 1$. After this operation is completed, there are $n - 1$ more operations that need to be scheduled on that machine, so $C_1 \geq t + n - 1$, where C_1 is the completion time of machine 1. When the final operation of machine 1 is completed, there are at least $m - 1$ more operations to be scheduled. So, $C_{max} \geq C_1 + m - 1$. Combining these inequalities shows that $C_{max} \geq (r - 1)m + (m - 1) + n = mr + n - 1$. ■

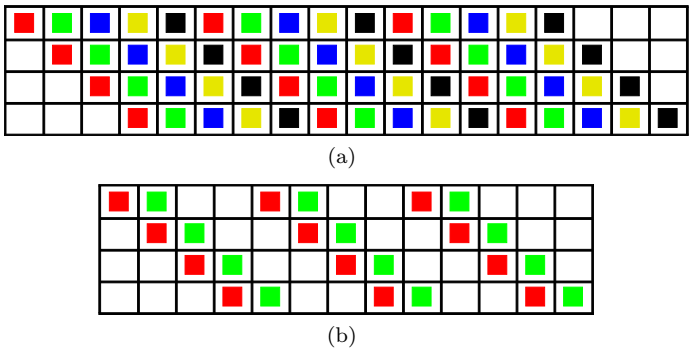


Fig. 6 An ObO schedule for unit-time cyclic-reentrant flowshops with $r = 3$ repetitions, $m = 4$ machines, and (a) $n = 5$ jobs ($n > m$) vs. (b) $n = 2$ jobs ($n < m$).

3.3 The Chain-reentrant Pattern

The chain-reentrant pattern $\phi = (1, 2, \ldots, m, 1)$, introduced by Wang, Sethi & van de Velde [23], consists of processing through all the machines in order, and finally returning to the first machine for the finishing operation.

An example of such pattern is the repairing of a vehicle in a garage. The visit in the garage begins with the quality control inspector. After that, the vehicle goes through a number of stations in a specified order, ending with a summary examination performed by the quality control inspector.

The chain-reentrant pattern appears in many realistic scenarios, such as signal processing, assembly of printed circuit boards, and wafer fabrication [23].

Denote by $Fm|chain\text{-}reentrant, p_{i,j} = 1|C_{max}$ a chain-reentrant flow shop with m machines, unit processing times, and makespan minimisation objective.

Theorem 3 *The ObO algorithm is optimal for* $Fm|chain\text{-}reentrant, p_{i,j} = 1|C_{max}$. *The makespan is given by* $C_{max} = \max\{n, m\} + n$.

Proof The proof is similar to that of Theorem 2 with machine 1 having two cycles and the other machines having only one cycle. We therefore omit the full proof and give only the highlights.

Case 1: $n \geq m$. When scheduling according to the ObO algorithm, each machine starts as early as possible and works continuously (Figure 7). Machine 1 ends last after two rounds, at time $2n$, so $C_{max} = 2n$.

Case 2: $n < m$. Machine 1 works with one break. It starts its second round at time m, and finishes working (with the last operation of the total schedule) after n more time units. Hence, $C_{max} = m + n$. ■

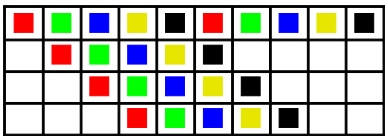


Fig. 7 An ObO schedule for a unit-time chain-reentrant flowshop with four machines and five jobs.

3.4 The Hub-reentrant Pattern

The hub-reentrant pattern $\phi = (1, 2, 1, 3, \ldots, 1, m, 1)$, introduced by Kubiak, Lou & Wang [12], consists of starting in the *hub* machine and visiting it alternately after each of the other machines in the production sequence.

An example of such pattern is the photolithography step in the wafer fabrication process of semiconductor production [6]. Photolithography is an optical process of etching multiple layers of circuit patterns on the silicon wafer. A wafer has to be processed at least ten times on the photolithography station, but between successive visits to that station, it has to be also processed on other machines.

Another example is of aircraft testing. Similarly to a car garage, every aircraft test starts and ends with a quality inspection test. In addition, after each activity, another quality control review is performed.

Denote by $Fm|hub\text{-}reentrant, p_{i,j} = 1|C_{max}$ a hub-reentrant flow shop with m machines, unit processing times, and makespan minimisation objective.

Theorem 4 *The ObO algorithm is optimal for* $Fm|hub\text{-}reentrant, p_{i,j} = 1|C_{max}$. *The makespan is given by* $C_{max} = nm$, *for* $n > 1$.

Proof When scheduling according to the ObO algorithm, machine 1 starts as early as possible, works continuously (for $n > 1$), and finishes last (Figure 8). Therefore, the schedule is optimal . Machine 1 ends last after m "rounds", at time mn, so $C_{max} = mn$. ■

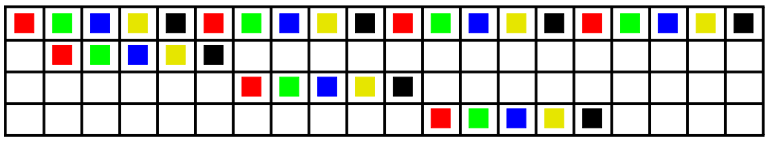


Fig. 8 An ObO schedule for a unit-time hub-reentrant flowshop with four machines and five jobs.

4 General Reentrant Patterns

In the previous section we showed that for the four discussed patterns, the problem $Fm|reentrant\ , p_{i,j} = 1|C_{max}$ is optimally solvable by the ObO al-

gorithm, which is greedy and polynomial (linear). For these patterns, the JbJ algorithm solves to optimality only some of the scenarios. In this section we regard the problem $Fm|reentrant\ , p_{i,j} = 1|C_{max}$, with general patterns, and show that the JbJ algorithm has practical significance when the number of jobs is large. Before, we present examples for showing that both the ObO and the JbJ algorithms are not generally optimal.

Example 3 $[n = 4, \phi = \{1, 2, 1, 3, 1, 4, 2, 4\}]$.

Figure 9 shows the JbJ and ObO schedules for Example 3. As evident, JbJ outperforms ObO. Thus, this contradicts the statements of Timkovsky [20,21] regarding the optimality of the ObO algorithm.

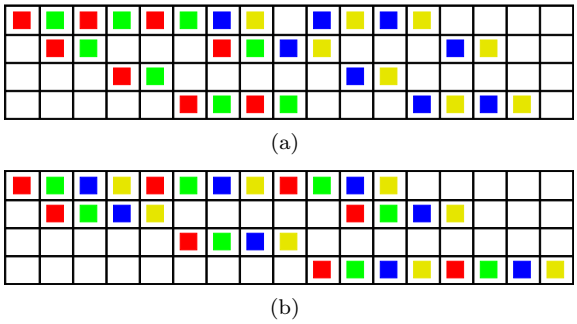


Fig. 9 Schedule of a reentrant flowshop with four jobs and pattern $\phi = \{1, 2, 1, 3, 1, 4, 2, 4\}$ (Example 3). (a) A JbJ schedule and (b) an ObO schedule.

Example 4 $[n = 3, \phi = \{1, 2, 1, 1, 2, 2\}]$.

In Example 4 neither ObO nor JbJ lead to an optimal solution. To show that, Figure 10 presents three schedules: optimal, ObO, and JbJ. Both the ObO and the JbJ schedules have a makespan of 13, while the optimal makespan is 11. Note that the optimal solution is obtained by a generalization of the ObO algorithm: the schedule is performed operation by operation, but consecutive operations that belong to the same machine are treated as one grouped operation.

The reported observation that $Fm|reentrant, p_{i,j} = 1|C_{max}$ is polynomial [20,21], is based on an incorrect statement regarding ObO being optimal. It follows that the complexity of the general problem (i.e., with general reentrant patterns) still needs to be determined.

Next we show that the JbJ algorithm, though not optimal, generally leads to a good schedule when the number of jobs is large. This observation is significant for a scenario of semi-mass production. Consider first the problem $Fm|reentrant, p_{i,j} = 1|C_{max}$, with a *balanced* reentrant pattern, i.e., a pattern such that each machine processes exactly the same number of operations per

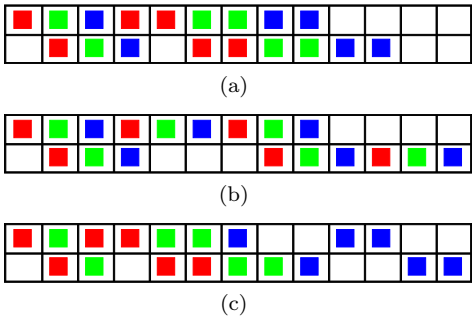


Fig. 10 Schedule of a reentrant flowshop with three jobs and pattern $\phi = (1, 2, 1, 1, 2, 2)$ (Example 4). (a) An optimal schedule, (b) an ObO schedule, and (c) a JbJ schedule.

job. By observing many examples, we have noticed that in the JbJ schedule, the machines work continuously most of the time. We give a more formal statement below, but first we show an example.

Example 5 [n jobs, $\phi = \{1, 2, 3, 1, 4, 4, 3, 2\}$].

The pattern of Example 5 is balanced, where each machine processes two unit-time operations per job. The JbJ makespan for $n = 100$ jobs is $C_{max} = 208$. Between times 17 and 197 all the machines work continuously. There are two periods, at the start and the end, for which not every machine works continuously. But, these periods do not scale with n. We observe that for general (not necessarily balanced) reentrant patterns, the JbJ algorithm returns a schedule in which the machines that process the maximal number of operations work continuously most of the time. We state this observation as a conjecture.

Definition 1 A machine that processes the maximal number of operations is called a *bottleneck machine*.

Definition 2 The maximal time for which a machine k works continuously in a given schedule is called the *central processing time*, and is dented by $\Delta(k)$.

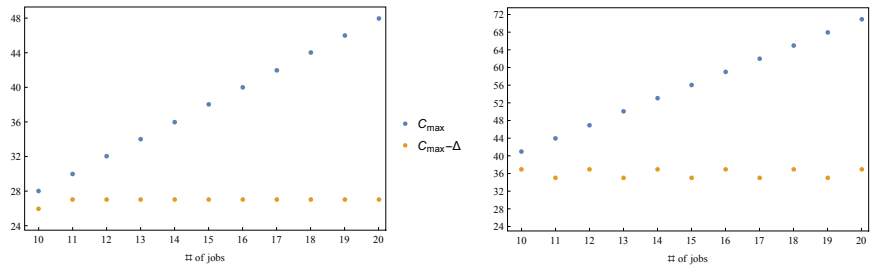
Conjecture 1 (Non-idle conjecture) The central processing time Δ of a bottleneck machine, in a JbJ schedule of $Fm|reentrant, p_{i,j} = 1|C_{max}$, is an increasing function of n; the time difference $C_{max} - \Delta$, may vary with n, but it is not a strictly increasing function of n. (Here, the notation C_{max} indicates $C_{max}(JbJ)$, i.e., the makespan of the JbJ schedule). From this it follows that $\lim_{n\to\infty} \frac{\Delta}{C_{max}} = 1$.

We give two examples of reentrant flow shop problems with four machines – the above mentioned Example 5 and Example 6 that follows. In both examples the reentrant pattern is balanced, i.e., all the machines are bottleneck machines processing 2 and 3 operations per machine, respectively.

Example 6 [n jobs, $\phi = \{1, 2, 3, 1, 1, 3, 4, 4, 4, 2, 2, 3\}$].

For Example 5 the time difference $C_{max} - \Delta$ of a JbJ schedule does not depend on n (for $n > 10$), as illustrated in Figure 11(a), showing the makesapn and the corresponding time difference as a function of n. Figure 11(b) shows the same variables for a JbJ schedule of Example 6. For this example, $C_{max} - \Delta$ does depend on n, but is not strictly increasing function of n (it is actually periodic in n, in accordance with the periodicity character that we describe next). According to Conjecture 1 this is a general feature of the JbJ schedule, for any given reentrant pattern. The conjecture is based on observations of many reentrant patterns, including non-balanced ones.

Corollary 1 *In case the non-idle conjecture is correct, it follows that* $\lim_{n\to\infty} \frac{C_{max}(JbJ)}{C_{max}(OPT)} = 1$, *where* $C_{max}(OPT)$ *is the makespan of an optimal schedule.*



(a) (b)

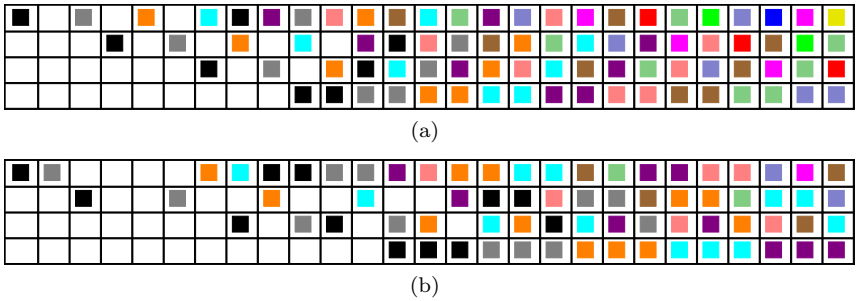
Fig. 11 C_{max} and $C_{max} - \Delta$ of a JbJ schedule as a function of n for (a) Example 5 and (b) Example 6.

Another observed characteristic of the JbJ schedule of a reentrant flow shop with a balanced pattern is that from a certain time, and continuing thereafter, all the starting jobs are arranged periodically. A *periodic* schedule is usually considered when there are several types of jobs, and each job should be repeated several (usually many) times. A schedule is called periodic[1] if it is composed of periodic repetition of batches of jobs. In a general problem, each batch is composed of all the job types, in the relative ratio needed for production; however, in our problem there is only one job type. Hence, in this study each batch is composed of identical jobs. The number of jobs in each batch, for a given periodic schedule, is denoted by q. The schedule cycle time C is defined by the periodicity relation $t_{i,s,r+1} = t_{i,s,r} + C$ where $t_{i,s,r}$ is the starting time of operation i ($i = 1, 2, \ldots, l$) of the s-th job ($s = 1, 2, \ldots, q$) in the r-th batch.

[1] The common term is "cyclic" but, as mentioned in Section 1, this has several meanings. To avoid confusion, we use the term "periodic" rather than"cyclic".

In Figure 12 we show a part of the JbJ schedule obtained for Examples 5 and 6, with a large number of jobs. To clarify the periodicity observation, jobs that have started prior to the presented starting time are not shown. Figure 12(a) shows the JbJ schedule of Example 5. Each batch is composed of only one job, having a production time of length 13, with a cycle time of $C = 2$. Figure 12(b) depicts the schedule of Example 6. Each batch contains two jobs, for example the black and grey coloured jobs. The first job in the batch lasts for 18 time units, the second lasts for 20 time units, and the cycle time is $C = 6$. In both examples, the number of operations of a bottleneck machine is equal to C. Hence, C is minimal, in accordance with the non-idle conjecture. The periodicity observations, exemplified in Figure 12, lead to the next conjecture.

Conjecture 2 (Periodicity conjecture) Consider a JbJ schedule of $Fm|reentrant, p_{i,j} = 1|C_{max}$ with a balanced reentrant pattern. There exists an index j^* such that disregarding all the jobs with $j < j^*$ gives a periodic schedule. According to Conjecture 1 this is an optimal schedule, i.e., the period C is minimal.



(a)

(b)

Fig. 12 Partial time-windows of the JbJ schedules for (a) Example 5 and (b) Example 6. Operations of jobs that started before the beginning of the time window are omitted.

5 Discussion

The focus of our study is a scheduling problem in a flow shop environment with three specifics: (1) reentrant, (2) identical jobs, and (3) unit-time.

There are few earlier papers that consider all three specifics. Tymkovsky conducted an extensive and impressive study over the reentrant flow shop problem and its complexity, including the problem we study in this article [20, 21]. While, unfortunately, he concluded that the ObO algorithm is optimal, we show that it is not. We also corrected the statements of Lev and Adiri regarding the JbJ algorithm being optimal for the V-reentrant pattern [13] and showed that the ObO algorithm is, in fact, optimal for that pattern. For

some other studied patterns the ObO algorithm is also optimal, which shows that the problem is polynomial, at least for these patterns. We did not find an algorithm that guarantees optimality for the general case, nor proved that the problem is NP-hard.

The article mainly deals with unit processing times. In this case, the obtained jobs are naturally identical jobs. Still, we have explicitly left the term "identical jobs" in the title. First, as a reminder of our main interest. The unit-time problem is a relatively simple starting point, but our interest is also in general processing times. Second, our result regarding the JbJ algorithm as being a good algorithm when the number of jobs is large, as reflected by Corollary 1, is also valid for general integral processing times if integral preemptions is allowed. This is because an operation with processing time p can be regarded as p unit-time consecutive operations. Therefore, $Fm|reentrant, p_{i,j} = p_i\ integral\ prmp|C_{max}$ is equivalent to the studied problem $Fm|reentrant, p_{i,j} = 1|C_{max}$.

We have presented the non-idle and periodicity conjectures; both are based only on studied examples. Obviously, examples do not prove that the conjectures are generally true. Yet, by observing many examples we suspect that the conjectures are indeed correct in general. The remaining open challenge is to prove the correctness of these conjectures. We note that even though the shown examples are of balanced patterns, the examples we have checked include also many non-balanced ones. The non-idle conjecture seems true for all patterns. For the periodicity conjecture the structure is more complex. In some non-balanced patterns the JbJ schedule is periodic. For example, the JbJ schedule of the pattern $\phi = \{1, 2, 1, 3, 1, 4, 4, 2, 4, 1\}$ is periodic with seven jobs per batch, and a cycle time of $C = 28$. Other pattern exists for which the JbJ schedule is not periodic. For example, for the pattern $\phi = \{1, 2, 3, 1, 3, 3, 4, 4, 2\}$ the periodicity relation is not valid for all the machines. From our observations the periodicity relation appears true at least for bottleneck machines. If this is indeed true than this proves Conjecture 2, since for a balanced pattern all machine are bottleneck machines. Clues for proving the conjectures may be found in the field of number theory, e.g., the Chinese remainder theorem [11].

Another challenge is in expanding our understanding towards more general problems that are closer to the original semi-mass production scenarios. In particular, finding algorithms that may be applied to the more flexible structure that was presented at the beginning of Section 2, like that shown in Example 1. In this article we have concentrated on one special case where any operation is processed by only one worker, i.e., the reentrant flow shop. Another special case that may inspire the more general scenario is that of *hybrid* flow shop[2], in which several workers can process a given operation [18,19]. The hybrid flow shop was studied also for identical jobs [3]. More flexibility properties are also discussed in the literature [7] and may be the basis for future research directions.

[2] Also termed flexible flow shop or multi-processor flow shop [17].

References

1. K1 A Aldakhilallah and R Ramesh. Cyclic scheduling heuristics for a re-entrant job shop manufacturing environment. *International Journal of Production Research*, 39(12):2635–2657, 2001.
2. Jen-Shiang Chen. A branch and bound procedure for the reentrant permutation flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 29(11-12):1186–1193, 2006.
3. Maged M Dessouky, Mohamed I Dessouky, and Sushil K Verma. Flowshop scheduling with identical jobs and uniform parallel machines. *European Journal of Operational Research*, 109(3):620–631, 1998.
4. Denise L Draper, Ari K Jonsson, David P Clements, and David E Joslin. Cyclic scheduling. In *IJCAI*, pages 1016 – 1021. Citeseer, 1999.
5. Claude R Duguay, Sylvain Landry, and Federico Pasin. From mass production to flexible/agile production. *International Journal of Operations & Production Management*, 17(12):1183–1195, 1997.
6. David J Elliott. Integrated circuit fabrication technology. 1982.
7. Hamilton Emmons and George Vairaktarakis. *Flow shop scheduling: theoretical results, algorithms, and applications*, volume 182. Springer Science & Business Media, 2012.
8. Kevin Forsberg and Harold Mooz. The relationship of systems engineering to the project cycle. *Engineering Management Journal*, 4(3):36–43, 1992.
9. Stephen C Graves, Harlan C Meal, Daniel Stefek, and Abdel Hamid Zeghmi. Scheduling of re-entrant flow shops. *Journal of Operations Management*, 3(4):197–207, 1983.
10. Nicholas G Hall, Tae-Eog Lee, and Marc E Posner. The complexity of cyclic shop scheduling problems. *Journal of Scheduling*, 5(4):307–327, 2002.
11. Thomas Koshy. *Elementary number theory with applications*. Academic press, 2002.
12. Wieslaw Kubiak, Sheldon XC Lou, and Yingmeng Wang. Mean flow time minimization in reentrant job shops with a hub. *Operations Research*, 44(5):764–776, 1996.
13. V Lev and I Adiri. V-shop scheduling. *European Journal of Operational Research*, 18(1):51–56, 1984.
14. Martin Middendorf and Vadim G Timkovsky. On scheduling cycle shops: Classification, complexity and approximation. *Journal of Scheduling*, 5(2):135–169, 2002.
15. JC-H Pan and Jen-Shiang Chen. Minimizing makespan in re-entrant permutation flowshops. *Journal of the Operational Research Society*, 54(6):642–653, 2003.
16. Jaroslaw Pempera and Czeslaw Smutnicki. Open shop cyclic scheduling. *European Journal of Operational Research*, 269(2):773 – 781, 2018.
17. Michael Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012.
18. Imma Ribas, Rainer Leisten, and Jose M Framiñan. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439–1454, 2010.
19. Ruben Ruiz and José Antonio Vázquez-Rodríguez. The hybrid flow shop scheduling problem. *European journal of operational research*, 205(1):1–18, 2010.
20. Vadim G Timkovsky. On the complexity of scheduling an arbitrary system. *Soviet Journal of Computer and System Sciences*, 5:46–52, 1985.
21. Vadim G Timkovsky. Cycle shop scheduling. In Leung Joseph YT, editor, *Handbook of Scheduling*, chapter 7. CRC press, 2004.
22. Chengen Wang, Chengbin Chu, and Jean-Marie Proth. Heuristic approaches for n/m/f/ ci scheduling problems. *European Journal of Operational Research*, 96(3):636–644, 1997.
23. Michael Yingmeng Wang, Suresh P Sethi, and Steef L van de Velde. Minimizing makespan in a class of reentrant shops. *Operations Research*, 45(5):702–712, 1997.
24. Xie Xie, Lixin Tang, and Yanping Li. Scheduling of a hub reentrant job shop to minimize makespan. *The International Journal of Advanced Manufacturing Technology*, 56(5-8):743–753, 2011.

MISTA 2019

A modeling methodology to support nurse rostering practitioners

Elín Björk Böðvarsdóttir · Pieter Smet · Greet Vanden Berghe · Thomas Stidsen

Abstract Researchers have studied the nurse rostering problem (NRP) for decades, producing many efficient algorithms for different formulations of the problem. In spite of these successes, the models and methods developed seldom make it to practical implementation. We believe too much emphasis has been put on solving the problem, and that the research community needs to take a step back and reflect on which problem they are solving. When modeling the NRP, we need to temper the violation of multiple constraints, and researchers tend to formulate the objective as minimizing the weighted sum of these violations. Although a solver may view a solution as optimal, practitioners might reject it if the *combination* of soft constraint violations is not acceptable. In this paper, we discuss some of the major drawbacks of this modeling approach and suggest a paradigm shift that may help us overcome them. We introduce the concepts *targets* and *acceptance thresholds* and discuss how we can integrate them into nurse rostering models and solution methods. We devote some attention to setting accurate acceptance thresholds, i.e., ensuring an *acceptable combination* of violations without the problem becoming infeasible. As this paradigm shift involves terms that practitioners can intuitively understand, it makes the automated scheduling approaches more accessible to them. Furthermore, the resulting approach can be employed for quality control, and we argue that it can yield more reliable rostering systems in practice.

Elín Björk Böðvarsdóttir
Department of Technology, Management and Economics, Technical University of Denmark, Anker Engelundsvej 1, 2800 Kgs. Lyngby, Denmark
E-mail: ebod@dtu.dk

Pieter Smet
KU Leuven, Department of Computer Science, CODeS, Gebroeders De Smetstraat 1, Gent 9000, Belgium
E-mail: pieter.smet@kuleuven.be

Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS, Gebroeders De Smetstraat 1, Gent 9000, Belgium
E-mail: greet.vandenberghe@cs.kuleuven.be

Thomas Stidsen
Department of Technology, Management and Economics, Technical University of Denmark, Anker Engelundsvej 1, 2800 Kgs. Lyngby, Denmark
E-mail: thst@dtu.dk

1 Introduction

The nurse rostering problem (NRP) is the task of assigning nurses to shifts to generate a feasible work schedule. This problem belongs to the class of personnel scheduling problems, which consider the assignments of employees to shifts for numerous different professions and scenarios. Even though we could extend this paper to consider personnel scheduling problems in general, the main focus will be on the nurse rostering problem for two reasons. First, it is one of the most widely studied subclass of personnel scheduling problems, and second, the problem requires round-the-clock scheduling, resulting in a more complicated set of constraints than some other types of personnel scheduling problems.

Researchers have studied the NRP for several decades and provided multiple formulations and solution methods (Burke et al, 2004; Van Den Bergh et al, 2013). In 2018, Kingston et al introduced a general format for expressing nurse rostering instances, extending a similar format for high school timetabling. They have converted several publicly available instances to this format, but we have yet to see how the research community will adopt to it.

Despite countless academic papers on nurse rostering, implementation in practice is limited. Kellogg and Walczak (2007) reported that only 30% of published research on nurse rostering had led to implementation, while 86% stated that they had intended for their work to be implemented. More recently, Petrovic (2019) reported that most of the available software for personnel scheduling had not benefited from the results of the research community. This gap from academic research to implementation in practice is intriguing, as the literature includes numerous examples of efficient solution approaches producing optimal or near-optimal solutions in a short time.

We believe that this lack of implementation is not due to our inability to solve the problem, but rather because we are not addressing the right problem. In an ideal world, we could generate perfect rosters with optimal staffing while fulfilling every single requirement for each nurse. However, in the real world, we cannot prevent all constraint violations, and thus, the challenge becomes to generate *high-quality* rosters with an *acceptable combination* of violations.

When Kellogg and Walczak (2007) analyzed the lack of implementation of nurse rostering research they named seven possible explanations, one of which being *lack of acceptance by nurses*. They related this lack of acceptance to two factors: the *lack of trust* in computerized systems and the *time required* to learn and use such systems.

Although this paper is over 10 years old and many have referenced Kellogg and Walczak (2007)'s results, no researchers have focused on the acceptance by nurses in their research since. Nonetheless, many have addressed the concept of *self-scheduling* (i.e., involving the nurses in creating their own schedules), which was another of Kellogg and Walczak (2007)'s possible explanations, along with *preference-scheduling*, which is closely related. Even though one can argue that having increased autonomy will likely lead to increased acceptance, this type of research addresses neither the lack of trust nor the time required. Only a single paper (Mihaylov et al, 2016) has addressed the time required, by introducing a method for automating a process that is currently manual.

In this paper, we discuss some major drawbacks with the current models in nurse rostering research (Section 2) and present a modeling methodology that is designed to simplify the information flow between the users and the solver (Section 3). Furthermore, we present preliminary results showing the computational potential of this methodology

for 12 real instances from Danish hospitals (Section 4). We believe that the increased transparency that this modeling methodology brings will both increase the trust in computerized rostering methods and reduce the time required to learn and utilize these methods.

2 Problem statement

Most often the NRP is formulated as a set of hard constraints and soft constraints, where the objective function minimizes a weighted sum of soft constraint violations. Despite this standard way of modeling the problem, the constraints that researchers choose to include may vary substantially. Furthermore, some researchers may formulate a given constraint as hard while other researchers argue that the same constraint should be soft, and in Kingston et al (2018)'s general format, each constraint can either be hard or soft, depending on the user's choice. This inconsistency is a clear indicator of how difficult it is to model this problem, as we need to capture different elements, many of which are conflicting.

Furthermore, practitioners are not interested in the objective value of a roster, and may even prefer suboptimal solutions if the combination of violations is better. Figure 1 provides an example, where we consider two soft constraints for overtime and float nurses (i.e., nurses not employed by the ward but hired in for single shifts as needed at an additional cost). The dashed lines indicate *acceptance thresholds*, i.e., the level of violations that practitioners will accept for each constraint. The objective function minimizes a weighted sum of these two constraints, and we indicate the decrease in the objective function along with displaying several feasible solutions marked with `x`. Although the one marked red is optimal within this set of solutions, practitioners would prefer one of the subpotimal solutions marked green as those are within the acceptance thresholds.

We argue that the complex nature of the weighted sum objective function is an important reason for the lack of successful implementation in practice. For the solution methods to generate high-quality rosters, accurate objective weights are essential (Burke et al, 2008). However, setting these weights is a difficult task, and Mihaylov et al (2016) revealed a substantial inconsistency between the weights that practitioners claim to be correct and their actions when manually generating rosters.

Both researchers and practitioners have raised concerns regarding this objective formulation and provided concrete examples of challenges related to choosing the weights (Gärtner et al, 2018; Petrovic and Vanden Berghe, 2012). When diving deeper into specific examples, we note that practitioners often increase weights in attempts to remove inevitable constraint violations, e.g., due to some hard constraints. This behavior is worrying, as it shows that practitioners are unaware of the types of solutions they can and cannot obtain. Moreover, one can see how these difficulties may lead to a very time-consuming process, while concurrently causing the rosters to be of unnecessarily poor quality.

Obtaining an in-depth understanding of the relationship between different constraints of the NRP is not trivial. For some pairs of constraints, we can intuitively predict the impact they have on each other, for example, that requiring a minimum number of days off does not support a constraint for a minimum number of workdays. Despite the conflict between these two constraints, we might be able to satisfy both depending on the bounds that have been set, along with how these two constraints

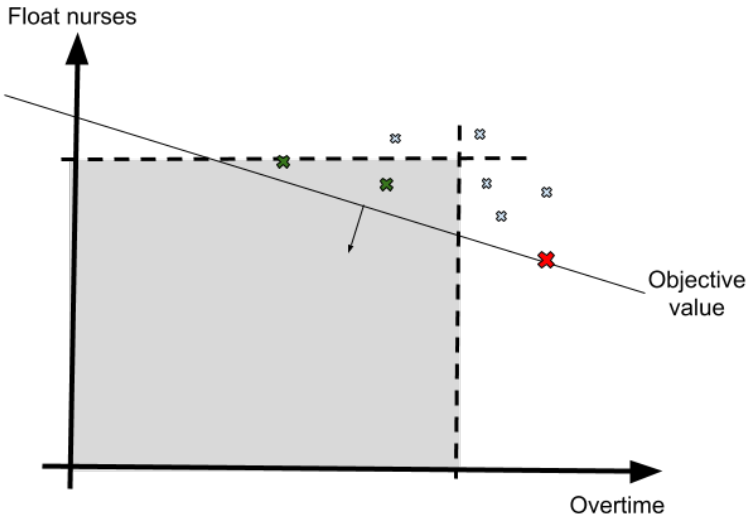


Fig. 1 Example showing how suboptimal solutions may include a better combination of violations.

interact with the remaining constraints in the formulation. Furthermore, the relation between two constraints might be more complicated than either supportive or conflicting, depending on the scenario. For example, a nurse's request for a day off can either support or conflict with a constraint for a minimum number of consecutive workdays, depending on where it is placed. Overall, the relationship between different constraints can be viewed as a spider web, i.e., a set of different threads tangled together, making it difficult to understand the role every single thread plays in the whole.

Due to the significant effect that poor rostering can have on nurses' health and happiness (Gärtner et al, 2018), the perceived quality of a roster is of utmost importance. Nonetheless, defining an overall measure for *quality* is impossible in this context, as many different factors impact it. By bundling everything up in a weighted sum objective, we lose control over individual components. As weights are subjective, a solution may be optimal (or near-optimal) while still including an unacceptable combination of constraint violations. Therefore, we stress the importance of addressing the right problem, not only by including relevant constraints but also by ensuring that they produce an acceptable roster.

3 Moving modeling towards a common language

Nurse rostering research could be strengthened with increased transparency and by simplifying the information flow between the user and the solver. Therefore, we adopt the shifted paradigm when formulating nurse rostering problems, from *objectives* and *objective weights* to *targets* and *acceptance thresholds*. For example instead of assigning a *weight* (penalty) to an *objective* for nurses working overtime, we instead set an *acceptance threshold* for an overtime-*target*, describing the amount we will accept.

This paradigm shift introduces some new challenges, namely how to set accurate acceptance thresholds for targets, and how to include these aspects in nurse rostering models and solution approaches. We will present a methodology to address these challenges in the following sections, with Section 3.1 focusing on the estimation of attainable acceptance thresholds and Section 3.2 focusing on how to apply these thresholds.

3.1 Estimating acceptance thresholds

When manually generating rosters, practitioners apply various rules of thumb, many of which could be translated into acceptance thresholds. Nonetheless, practitioners often need to bend these rules to generate feasible rosters, and one may even see that the *hardest* of constraints become violated under some circumstances. Thus, finding universal values for these acceptance thresholds is unrealistic. Instead, we should focus on deriving attainable acceptance thresholds based on the instance at hand.

Acceptance thresholds are related to bounds on the number of violations of soft constraints. For example, for a constraint k with a bound of n_k violations, a corresponding acceptance threshold could be accepting $n_k + \delta_k$ violations. In this case, δ_k represents some slack that is intended to ensure that a feasible solution exists. Nonetheless, estimating accurate bounds presents a challenge as the acceptance thresholds should neither be too tight (causing infeasibility) nor too loose (unnecessarily compromising the quality of the rosters).

An important early work in this area is by Burke et al (2002), where they used bounds for soft constraint violations to introduce an alternative objective function formulation. They estimated the bounds for each constraint separately, where the lower bound represented the lowest possible number of violations. They mapped the lower bounds to an *ideal point* in the search space, and acknowledged that it would likely lie in an infeasible part of the solution space. Nevertheless, allowing an appropriate slack from the ideal point may translate into attainable acceptance thresholds.

When estimating acceptance thresholds, we should not only consider a single constraint. Instead, we should pragmatically choose relevant components of the problem. For example, considering float nurses and overtime separately will not produce reliable bounds as they complement one another in a roster. As setting acceptance thresholds for multiple constraints simultaneously often becomes a question of trade-off, we need to incorporate practitioners' priorities in the procedure. Theoretically, we could assign multiple float nurses to completely avoid overtime, or vice versa, meeting the staffing requirements without float nurses by assigning the nurses to significant overtime. These examples represent the two extremes, but in practice the desired outcome lies somewhere between. Table 1 provides an example where the first two rows represent the two extremes, while the remaining rows present some different combinations of acceptance thresholds. The accurate combination depends on each instance, for example, the ward's budget for float nurses or how well the overtime spreads among different nurses.

Estimating bounds for a component of the problem is closely related to the concept of *minimal unsatisfiable subsets* (i.e., a set of constraints such that no feasible solution exists but all strict subsets have a feasible solution). When solving a minimization problem with several terms in the objective function, we obtain a feasible *combination* of bounds, given the objective weights (similar to those presented in Table 1). Reformulating these objective terms as hard constraints, using the combination of bounds

Table 1 Different acceptance thresholds for float nurses and overtime, depending on practitioners priorities.

Float nurses	Overtime [in hours]
10	0
0	70
5	30
3	50
8	10

as an upper bound for each term, would yield the same result. However, if we would make one of the bounds tighter (without changing other bounds) it would result in an infeasible problem, where the corresponding constraint would become unsatisfiable. Thus, we need to set the acceptance thresholds for relevant components of the problem simultaneously to avoid unsatisfiable subsets.

In some cases, acceptance thresholds may correspond to organizational targets, e.g., fulfilling a minimum percentage of requests for days off. Nonetheless, we might need to relax this minimum if infeasible for the instance considered. Thus, we may base some acceptance thresholds on rules of thumb, but meanwhile, we need to ensure that those rules are adapted to each instance as needed.

The following sections present two examples exhibiting the methodology for adapting organizational targets to a specific instance, resulting in attainable acceptance thresholds. The former example considers only a single target, while the latter considers three targets simultaneously.

3.1.1 Maximum number of consecutive workdays

To show how we can adapt acceptance thresholds to a specific instance we consider the maximum number of consecutive workdays. This is a common constraint in many personnel scheduling problems, as it is often imposed by legislation, for example, that there should at most be n consecutive workdays. Nonetheless, this constraint is not always formulated as a hard constraint as legislation may leave room for exceptions, to better accommodate the employees' individuality. Thus, the objective function often includes a term that penalizes for exceeding n consecutive workdays.

Algorithm 1 displays how to extend the general threshold n for nurse i to day-specific acceptance thresholds n_d^i for all days of the rostering horizon D. In line 2, we consider when we should assign the next day off. When generating rosters in practice, the nurses are often involved in the process, e.g., by requesting which days they would and would not like to work. Furthermore, some shift assignments may be fixed beforehand (e.g., due to administrative meetings) and the feasibility of scheduling a day off may depend on these assignments. Finally, we emphasize that this estimation does not consider any other constraints of the problem, but is solely based on pre-processing the input data for the nurse in question.

This extension may be used to manage where these type of constraint violations occur, i.e., only when an employee has specifically allowed ignoring this constraint. Although the number of violations of the general threshold might be the same (and thus considered equally good by the objective function), the combination of violations would be more acceptable. Estimating the acceptance thresholds in this manner can

Algorithm 1 Adapting the acceptance threshold for the maximum number of consecutive workdays to a specific nurse i.

1: **for** $d \in D$ **do**
2: Estimate $\mu_d :=$ the number of days from d until a day off is feasible and the employee has not specifically requested a work assignment
3: Set $n_d^i = \max(n, \mu_d)$ as the acceptance threshold for number of consecutive workdays from day d for nurse i
4: **end for**

be highly valuable in practice, as it enables adapting general rules to the uniqueness of individual employees.

3.1.2 Available resources

This section indicates how we can estimate acceptance thresholds for a group of constraints simultaneously. We provide two examples of how to control the available resources in a rostering environment where the nurses have the opportunity to influence their rosters, similar to self-scheduling. If we assume that the staffing requirements are hard constraints, then the available resources refer to the ward's nurses along with float nurses needed to fulfill the minimum staffing. We would like to minimize the deviation from the nurses' contractual hours, corresponding to two targets: one for *overtime* and another for *undertime*. Additionally, we have a third target as we would like to minimize the number of *float nurses* needed to meet the staffing requirements.

Algorithm 2 displays how to employ a nurse rostering model along with any optimization method to estimate bounds for the targets, and then translate these bounds into acceptance thresholds. In line 3, we let the objective weights represent the priorities of these targets, ensuring that those priorities are transferred into the acceptance thresholds. The results are not very sensitive when it comes to the weights, and preliminary results have shown that setting the weight for float nurses to 10 and the other two to 1 produces accurate bounds (assuming that all objectives penalize based on the same measurement unit). When translating the bounds n_k into acceptance thresholds λ_k we add a small slack δ_k, that should give the flexibility to ensure feasibility w.r.t. other targets.

Algorithm 2 Estimating bounds for available resources using any solver.

1: Consider a full nurse rostering model
2: Deactivate objectives not directly corresponding to these targets
3: Set the objective weights for the corresponding targets to represent the priorities
4: Optimize to obtain bounds n_{float}, n_{over}, n_{under}
5: Add appropriate slack to obtain acceptance thresholds
$\lambda_k = n_k + \delta_k$ for $k \in \{float, over, under\}$

Instead of using a full nurse rostering model to obtain these bounds, we can also analyze the rostering data prior to the optimization to derive acceptance thresholds. We assume some general thresholds have been set, e.g., employing zero float nurses and allowing a deviation from contractual hours corresponding to half a shift. After analyzing the data we loosen these general thresholds to ensure feasibility.

Algorithms 3-4 display how to derive nurse-specific acceptance thresholds for overtime and undertime, respectively. Mainly, two factors control whether overtime or undertime is necessary, namely the surplus or deficit from the previous rostering horizon along with the shifts and days off that have already been fixed with self-scheduling.

In Algorithm 4 (line 1) we consider the days that have not yet been fixed during self-scheduling and estimate a lower bound on the number of working hours that we can assign on those days. In this estimation we take into account various factors concerning the already fixed days, including the number of such days and whether they include work or a day off. Furthermore, we distinguish between weekdays and weekends, as we often have organizational constraints that restrict weekend work.

Algorithm 3 Deriving acceptance thresholds for overtime for a single nurse i based on the specific instance.

1: Calculate `Fixed working hours` from input data
2: `Minimum assigned hours` = `Fixed working hours` + `Surplus from previous horizon`
3: **if** `Minimum assigned hours` > `Contractual hours` + `General threshold` **then**
4: λ^i_{over} = `Minimum assigned hours - Contractual hours`
5: **else**
6: λ^i_{over} = `General threshold`
7: **end if**

Algorithm 4 Deriving acceptance thresholds for undertime for a single nurse i based on the specific instance.

1: Estimate `Lower bound for the addition` to the hours that have already been fixed
2: `Possible hours` = `Fixed working hours` + `Lower bound for the addition`
3: **if** `Possible hours - Deficit from previous horizon` < `Contractual hours - General threshold` **then**
4: λ^i_{under} = `Contractual hours - Possible hours`
5: **else**
6: λ^i_{under} = `General threshold`
7: **end if**

Algorithm 5 displays how we estimate the acceptance threshold for float nurses in three steps. First, we consider each specific coverage constraint, composed of a day, a shift and a skill (lines 1-5). Second, we consider the accumulated coverage required for all shift and skill combinations on a specific day (lines 6-12). Third, we consider the accumulated coverage required during the entire horizon for a specific shift and skill combination (lines 13-20).

In the third step (line 14), we estimate an upper bound on the total number of times we can assign a nurse of a given skill to a given shift. This estimation is based on either a fixed maximum for how often each nurse can be assigned to the specific shift or calculated based on the `Possible hours` (from Algorithm 4). Thus, an inaccurate estimation for one threshold may impact the estimate for other thresholds. Therefore, we need to identify which targets are related and take it into consideration when estimating the thresholds.

Algorithm 5 Deriving acceptance threshold for float nurses based on the specific instance.

```
for coverage constraints do
    if the number of available nurses for the shift and skill on the day is less than the required coverage then
        Increase acceptance threshold for float nurses
    end if
end for
for days of the rostering horizon do
    if the total number of available nurses is less than the accumulated coverage needed for the day then
        if this difference does not arise due to a specific combination then
            Increase acceptance threshold for float nurses
        end if
    end if
end for
for (shift,skill)-combinations do
    Estimate upper bound for the number of nurses with skill assigned to shift during entire rostering horizon
    if the upper bound is lower than the total required coverage then
        if this difference has not been considered on a specific day then
            Increase acceptance threshold for float nurses
        end if
    end if
end for
```

3.2 Applying acceptance thresholds

By estimating attainable acceptance thresholds, we provide practitioners with beneficial guidelines on the types of solution that they can obtain. When considering the common weighted sum objective function, these thresholds support the practitioners in making educated decisions when setting the weights. Nonetheless, we gain even more by incorporating acceptance thresholds directly into automated scheduling methods. Unfortunately, this is not trivial and we will discuss some of the new challenges that it may bring.

If we are certain that at least one feasible solution complying with all acceptance thresholds exists, then an obvious approach would be to enforce the acceptance thresholds as hard constraints in the formulation. However, one needs to tread carefully, as claiming to have certainty when considering all future rostering horizons is rather unreasonable. Thus, we need to introduce the acceptance thresholds to the formulations without introducing new challenges concerning infeasibility.

Even though obtaining an intuitive understanding of objective weights is difficult, they currently play an important role in trying to capture the priorities of different constraints. Thus, future research should investigate simpler alternatives for capturing these priorities. Another crucial question is how to directly include acceptance thresholds in nurse rostering models while ensuring to avoid infeasibility. Obtaining answers to these questions will aid in simplifying the objective function, resulting in nurse rostering models becoming more accessible for practitioners.

In the following sections, we present two methodologies for employing targets and acceptance thresholds in automatic scheduling. The former is a supplement for any model with a weighted sum objective, while the latter is a reformulation of the problem, which removes the complexity of the weights.

3.2.1 Weight Tuning

Although objective weights are an abstract concept, they are still used in a far majority of nurse rostering formulations. We believe that making weighted sum formulations more accessible for practitioners can increase the implementation of already developed nurse rostering approaches. Therefore, we introduce a general methodology, *Behind-the-Scenes Weight Tuning*, that employs targets and acceptance thresholds to automatically set the weights when working with a weighted sum objective.

Starting from an initial set of weights, the methodology evaluates the quality of the resulting roster by comparing the constraint violations to the corresponding targets and their acceptance thresholds. If the roster does not reach some target(s), we automatically adjust the supporting weight(s) to penalize such violations more heavily. With the new set of weights, we re-optimize and perform the same evaluation and (eventual) adjustments to further improve the perceived quality of the roster.

This methodology does not alter the complex objective function but instead provides automatic assistance with the associated challenges. While the set of feasible solutions remains the same, *Behind-the-Scenes Weight Tuning* guides the optimization method to more promising solutions by altering the objective weights. The methodology is independent of the formulation and functions as a supplement on current nurse rostering models and solution methods. Thus, it is ideal to facilitate the paradigm shift.

3.2.2 Goal programming

The concepts of targets and acceptance thresholds fit quite naturally in the framework of goal programming. For every target, we should stay within the corresponding acceptance threshold if feasible, and if not, then we should minimize the deviation from the threshold. As a standard goal programming reformulation has everything bundled together in a single objective, we would still need the complex concept of objective weights to capture the priorities of different targets.

Thus, we suggest reformulating the problem as a lexicographic goal programming model, with a distinct priority for each target. For each target, we solve a simple goal program, where the only objective is to minimize the deviation from the corresponding acceptance threshold. If the objective value is zero, then we have a feasible solution satisfying the acceptance threshold and may introduce it as a hard constraint. If the objective value is greater, then we must adjust the acceptance threshold accordingly before introducing it as a hard constraint.

With this approach, we iteratively introduce the acceptance thresholds as hard constraints, but only after confirming their feasibility. Furthermore, when reaching a target that cannot comply with the threshold, we adapt the threshold knowing that it will not be feasible without violating the acceptance thresholds for targets of a higher priority.

Figure 2 presents how to apply this methodology to a small nurse rostering example, where we have four targets and the only hard constraints relate to coverage. The targets (in prioritized order) correspond to the number of float nurses, exceeding the maximum number of consecutive workdays, the amount of overtime and the amount of undertime. As shown, we move through the targets in prioritized order and introduce them as hard constraints once we have verified their feasibility. If we need to adjust a threshold, the appropriate adjustment comes directly from the value of the objective function and

thus, we do not need to reiterate in any step. When solving the goal program for the last target we obtain an optimal roster w.r.t. the priorities and the original acceptance thresholds.

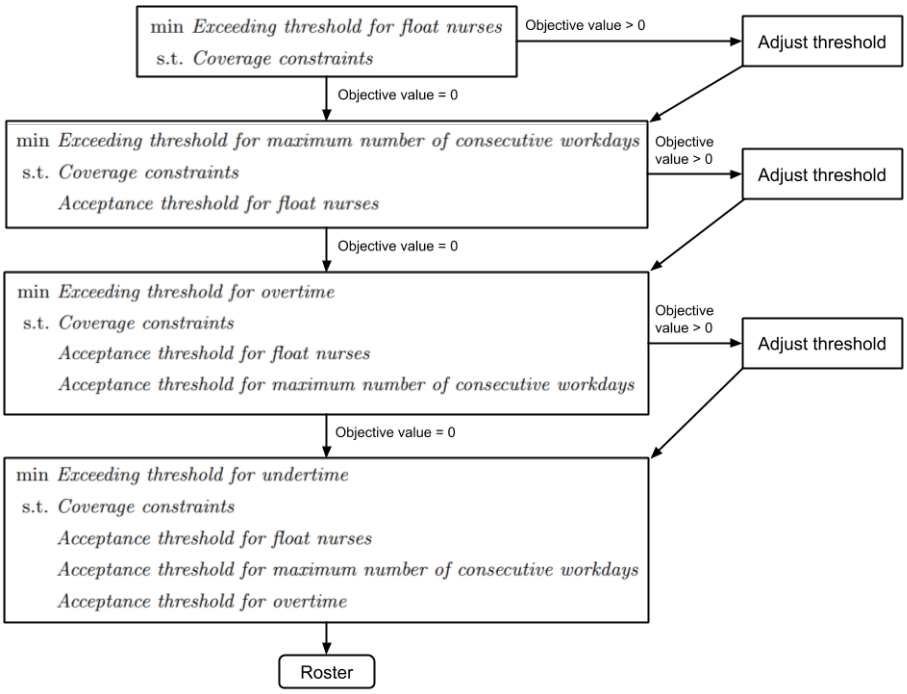


Fig. 2 Example of the lexicographic goal programming methodology.

The most valuable advantage from working with accurate acceptance thresholds is that we can exclude targets of a lower priority when optimizing each target, as they have implicitly been incorporated in the acceptance thresholds. For comparison, if we use a goal of zero violations at each level, the subsequent hard constraints would presumably be too tight to enforce them at lower priority levels. As an example, when minimizing the number of float nurses considering only the coverage constraints, we may obtain a solution that uses no float nurses. However, when introducing the remaining targets, we will need five float nurses to arrive at a reasonable roster. If we possess this knowledge beforehand, we can set the initial acceptance threshold (and thus the bound on the corresponding hard constraint) accordingly. However, without the knowledge we would enforce a hard constraint forbidding all float nurses, resulting in obscure violations for other targets when moving down the prioritized list.

4 Preliminary results

This section includes some preliminary results for the estimation techniques described in Section 3.1. We provide results for 12 real-world nurse rostering instances published

by Böðvarsdóttir et al (2019b), that have a rostering horizon of 28 days and include 40-50 nurses.

We consider the four targets presented throughout this paper, with the following priority ordering: the number of float nurses, exceeding the maximum number of consecutive workdays, the amount of overtime and the amount of undertime. In addition to these targets, we include numerous hard constraints, for example, coverage constraints and a maximum number of assignments to specific shifts. A complete overview of the hard constraints is presented by Böðvarsdóttir et al (2019a).

We set the general acceptance thresholds to zero for the number of float nurses, while the general threshold for overtime and undertime is 5 hours, corresponding to roughly half a shift. Furthermore, we have a fixed maximum on the number of consecutive workdays, and set a general threshold for never exceeding this maximum. We adapt these acceptance thresholds to each instance using Algorithm 1 (Section 3.1.1) and Algorithms 3-5 (Section 3.1.2).

Table 2 shows the violation of these thresholds when solving the problem using the lexicographic goal programming methodology. For 11 out of the 12 instances, the acceptance thresholds for all four targets are transformed into feasible hard constraints, and for the remaining instance, we only need to extend the acceptance threshold for a single target. These results clearly exhibit how relatively simple pre-processing techniques can yield attainable acceptance thresholds.

For instance B04, we need to extend the acceptance threshold for overtime by almost nine hours in total, distributed among two nurses. The main reason for this extension is a conflict between the targets for float nurses and overtime on a single day. Thus, we assign the two nurses to work during the day to avoid employing float nurses, resulting in the overall working time for these nurses exceeding their requested overtime. By analyzing the roster that was employed in practice, we can confirm that the individual rosters for these two nurses are identical, meaning that the managers accepted the extension of this threshold.

Table 2 The violations obtained with the lexicographic goal programming framework.

Instance	Float nurses [number]	Maximum number of consecutive workdays [days]	Overtime [hours]	Undertime [hours]
A01	0	0	0	0
A02	0	0	0	0
A03	0	0	0	0
A04	0	0	0	0
A05	0	0	0	0
A06	0	0	0	0
A07	0	0	0	0
B01	0	0	0	0
B02	0	0	0	0
B03	0	0	0	0
B04	0	0	8.6	0
B05	0	0	0	0

Accurate acceptance thresholds should not only be feasible but also sufficiently tight to prevent us from accepting a solution with an unnecessarily high number of violations. We estimate the slack for the different targets using Equation (1) for the

number of float nurses and Equation (2) for the remaining targets, as the acceptance thresholds for those are nurse-specific. Table 3 displays the unused slack for the four targets.

$$\frac{\text{Acceptance threshold} - \text{Amount assigned}}{\text{Acceptance threshold}} \tag{1}$$

$$\frac{\sum_{\text{nurses}}(\text{Extension of general threshold} - \text{Use of extension})}{\text{Number of nurses with extension}} \tag{2}$$

Table 3 The unused slack when extending the general acceptance thresholds. A dash (-) indicates that the acceptance threshold was too tight (not feasible).

Instance	Float nurses [%]	Maximum number of consecutive workdays [days/nurse]	Overtime [hours/nurse]	Undertime [hours/nurse]
A01	0	0.36	0.00	5.28
A02	0	2.07	0.00	3.50
A03	0	3.00	1.00	10.19
A04	0	1.24	7.60	6.05
A05	0	0.44	0.94	2.50
A06	0	2.67	1.00	42.41
A07	0	1.15	1.00	0.00
B01	0	4.00	0.00	13.16
B02	0	1.21	4.00	34.85
B03	0	1.33	1.00	56.23
B04	0	1.75	-	9.03
B05	0	2.67	1.00	13.75

The acceptance thresholds for float nurses are accurate, as the number of float nurses we assign when employing the lexicographic goal programming methodology is exactly equal to the estimated acceptance threshold for all 12 instances.

The unused slack for exceeding the maximum number of consecutive workdays differs between the instances, ranging from almost no slack and up to a slack of 4 days per nurse. As we only extend the general threshold for specific days according to the nurses' requests, we will not see any unwanted violations of the general threshold even though the acceptance threshold is rather loose. Thus, having additional slack for this target is not a cause for worry.

The extension of the acceptance threshold for overtime is tight, with the unused slack being at most 1 hour for a majority of the instances. The two exceptions (A04 and B02) both include the Christmas holidays, where a part of the roster was fixed by the manager due to individual agreements with the nurses, resulting in more `Fixed working hours` (Algorithm 3, line 2). Furthermore, we must emphasize that the slack for all instances is less than the length of a single shift.

The unused slack for undertime is below two shifts per nurse for 75% of the instances. For the three remaining instances, the slack is quite substantial or ranging from 4 to 7 shifts per nurse. For all those instances, one or more nurses have significant undertime at the start of a rostering horizon, as a result of working too few shifts in previous horizons. As we cannot guarantee that we will be able to correct for the original deviation we need to set the acceptance thresholds accordingly. As a result, we see a lot of unused slack for nurses where we can reduce the deviation to some extent.

Overall, the thresholds for undertime were intentionally set to be sufficiently loose, as the undertime target conflicts with numerous other constraints and targets. For example, we may need increased undertime to ensure that we do not exceed the maximum number of consecutive workdays, along with ensuring a sufficient number of days off in general. Thus, when estimating the `Possible hours` (Algorithm 4, line 1) we must avoid too tight of a bound to ensure an overall feasible roster exists for the nurse. In many cases, we will never reach the acceptance threshold, because other constraints (such as coverage constraints) prevent such amount of undertime from being feasible for the problem as a whole (explaining the substantial slack in Table 3).

In general, the tightness of each acceptance threshold differs from instance to instance, and from employee to employee. Nonetheless, we confidently state that none of the acceptance thresholds are too loose, meaning that we will not see a practitioner rejecting a solution because of inaccurate thresholds.

5 Conclusion

In spite of a great deal of nurse rostering research throughout the years, the advances made in the research community rarely reach implementation in practice. The complexity of the current formulations is beyond the understanding of most practitioners, making them resistant to adopt them. Furthermore, an overall estimation of quality using a single objective function is far from the intricate decisions that practitioners face.

We have presented a shift in paradigm for nurse rostering research, designed to address these major disadvantages of the current state-of-the-art. This shift puts increased emphasis on the desired outcome, not thinking about the objective function's details but rather about what it can produce. This thought process is essential when working with personnel scheduling and other similar problems, where the choice of objective is debatable. At the conference, we will discuss the implications of working with targets and acceptance thresholds and present computational evidence of its potential for real-world instances.

We see many advantages arising from this modeling methodology. By adaptively setting acceptance thresholds we can better accommodate the unique needs and preferences of individual employees. Furthermore, providing practitioners with information about attainable acceptance thresholds is highly valuable, as we have seen that they frequently try to resolve inevitable constraint violations. Additionally, with accurate acceptance thresholds, we have an opportunity to increasingly automatize the process, reducing the overall need for manual interventions.

By moving away from the abstract concept of objective weights and determining attainable acceptance thresholds, we significantly increase the transparency. As we are moving towards a language that all can understand, we give practitioners a greater opportunity to understand and trust the models. Finally, by simplifying the information flow, we eliminate some critical challenges that practitioners have faced and thus reduce the time and effort needed to generate rosters. Therefore, this shift in paradigm can aid in increasing acceptance by practitioners, conceivably resulting in an increased adaption and implementation in practice.

Acknowledgements We thank the practitioners we have worked with, along with all practitioners that have collaborated with researchers throughout the years to bridge the gap from

research to practice. Furthermore, Elín Björk Böðvarsdóttir acknowledges Data and Development Support at Region Zealand, Denmark, for financial support and collaboration in her PhD project. Research partially supported by Data-driven logistics (FWO-S007318N).

References

Burke EK, De Causmaecker P, Petrovic S, Vanden Berghe G (2002) A multi criteria meta-heuristic approach to nurse rostering. Proceedings of the 2002 Congress on Evolutionary Computation, Cec 2002 2:1004,413, 1197–1202, DOI 10.1109/CEC.2002.1004413

Burke EK, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. Journal of Scheduling 7(6):441–449, DOI 10.1023/B:JOSH.0000046076.75950.0b

Burke EK, Curtois T, Post G, Qu R, Veltman B (2008) A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operational Research 188(2):330–341

Böðvarsdóttir EB, Bagger NCF, Høffner LE, Stidsen T (2019a) A comprehensive integer programming formulation of the nurse rostering problem in Denmark. Technical report

Böðvarsdóttir EB, Bagger NCF, Høffner LE, Stidsen T (2019b) Data for research on nurse rostering in Denmark [Data set]. https://doi.org/10.5281/zenodo.3374636

Gärtner J, Bohle P, Arlinghaus A, Schafhauser W, Krennwallner T, Widl M (2018) Scheduling matters-some potential requirements for future rostering competitions from a practitioner's view. In: PATAT 2018 - Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling, pp 33–42

Kellogg DL, Walczak S (2007) Nurse scheduling: From academia to implementation or not? Interfaces 37(4):353–369, DOI 10.1287/inte.1060.0247

Kingston JH, Post G, Vanden Berghe G (2018) A unified nurse rostering model based on xhstt. In: PATAT 2018 twelfth international conference on the practice and theory of automated timetabling, Vienna, August, pp 81–96

Mihaylov M, Smet P, Van Den Noortgate W, Vanden Berghe G (2016) Facilitating the transition from manual to automated nurse rostering. Health Systems 5(2):120–131

Petrovic S (2019) "You have to get wet to learn how to swim" applied to bridging the gap between research into personnel scheduling and its implementation in practice. Annals of Operations Research 275(1):161–179

Petrovic S, Vanden Berghe G (2012) A comparison of two approaches to nurse rostering problems. Annals of Operations Research 194(1):365–384, DOI 10.1007/s10479-010-0808-9

Van Den Bergh J, Beliën J, De Bruecker P, Demeulemeester E, De Boeck L (2013) Personnel scheduling: A literature review. European Journal of Operational Research 226(3):367–385, DOI 10.1016/j.ejor.2012.11.029

MISTA 2019

Production Planning based on Supply Chain Volatility Clustering: A Machine Learning Approach

Shusheng Ding · Tianxiang Cui · Ruibin Bai

Received: date / Accepted: date

Abstract Market uncertainties play vital roles in supply chain management, and dynamic economic environment puts us into a turbulence episode for managing the supply chain. This paper mainly focuses on market uncertainties reduction to formulate production plan for manufacturing companies. The production plan based on volatility clustering model would be valuable in the reduction of market uncertainties. We have used two volatility clustering models, one is the GARCH model and the other is the machine learning model called Density Based Spatial Clustering of Applications with Noise (DBSCAN). Comparing the performance of the two models, we unveil that the performance of machine learning model overwhelmingly outperforms the GARCH model. We contend that correlations between two time series and noise of model estimation play crucial roles in volatility clustering and the machine learning model has taken those two factors into account. As a consequence, our result implies that manufacturing companies production plan can be strengthened through the market uncertainties reduction by using DBSCAN.

Keywords supply chain management; volatility clustering; machine learning; DBSCAN

Shusheng Ding
School of Business, Ningbo University

Tianxiang Cui* (corresponding author)
School of Computer Science, The University of Nottingham Ningbo China
E-mail: zx10551@nottingham.edu.cn

Ruibin Bai
School of Computer Science, The University of Nottingham Ningbo China

1 Introduction

Supply Chain Management (SCM) plays an important role in manufacturing companies in terms of resource allocation and cost minimization (Olhager and Selldin, 2004 [19]). Likewise, Narasimhan and Jayaram (1998 [18]) show that an integrated supply chain management can achieve superior manufacturing performance. Dehning et al., (2007 [6]) also reveal that the adoption of high-technology based supply chain management would be tremendously helping for manufacturing companies to enhance firm performance. More recently, emerging type of supply chain management such as green supply chain management is widely applied in Canadian and Chinese manufacturing companies (see Li et al., 2016 [15], Kaur et al., 2018 [14]). As a result, companies especially manufacturing companies value SCM highly.

However, sound supply chain and logistics management is challenged by risk factors and environmental uncertainties (Aqlan and Lam, 2016 [2]). As a result, the ability to respond to those uncertainties can assist companies to strengthen their core competence and alleviate its long-term business sufferings. (Hendricks and Singhal, 2005 [11]). Therefore, the supply chain decision making framework which takes uncertainty into account in order to reduce cost becomes largely demanding in recent years (Guerra-Zubiaga and Young, 2008 [10]).

As a result, the main contribution of this paper is to develop a framework for manufacturing companies to formulate their production plans via the reduction of market uncertainties. Our framework is based on the volatility clustering model and we schedule all market related activities such as raw material purchase and product distribution into low volatility periods. Under such framework, we can thereby reduce the overall market price uncertainty faced by manufacturing companies as all market price uncertainty can be reduced through rescheduling. We adopt a decision tree method to determine the next production stage in the supply chain based on the volatility clustering result.

Empirically, we use four key business variables to extract market price uncertainty information mentioned in Christopher and Holweg (2011 [5]), which are heating oil price, copper price, Baltic Dry Index (BDI) and exchange rate. For heating oil price and copper price, we take the price information from Chinese commodity markets and for exchange rate, we take the Renminbi (BMB) against Euro. These four key business variables represent the raw material cost, the transportation cost and the sale price of exportation, respectively. We adopt the GARCH (generalized autoregressive conditional heteroscedasticity) model for the conditional volatility estimation and volatility clustering. It has been well documented that GARCH estimation of volatility is consistent with volatility clustering patterns exhibited in financial time series (see Bai et al., 2003 [4], Alberg et al., 2008 [1], Ardia et al., 2019 [3]).

Based on the data of four key business variables, we deliver empirical results to reveal that our framework can aid manufacturing companies to save operating costs through decreasing market uncertainty. We firstly discover that the GARCH model has a negative improvement for the cost saving side

Key Business Variables	Obs	Mean	Std. Dev.	Min	Max
Heating Oil	3,023	7.34	0.79	5.67	9.37
Copper	3,023	8.24	0.23	7.51	8.63
BDI	3,023	10.83	0.21	10.01	11.24
Exchange Rate	3,023	6.74	0.13	6.47	7.01

Table 1 Statistical summary of four key business variables.

of production plan. The GARCH model delivers as much as -66% of the overall improvement rate. Then, we change to the machine learning model and the model performance has been improved. For the heating oil and copper purchase cost, our machine learning model can save as much as 6.22% and 9.15% cost compared with benchmark model without scheduling. For transportation cost, our framework can help manufacturing companies to save 14.11% cost. For the sale price of export, our framework can raise the revenue for sale around 1.5%. The overall cost saving and revenue generation of our machine learning model through production scheduling is around 30%. Therefore, it is arguable that our machine learning framework would be helpful to manufacturing companies for cost saving, especially in the volatile periods.

The remainder of the paper is organized as follows. Section 2 covers data description of key variables including heating oil price, copper price, BDI and exchange rate as well as the GARCH type model setup. Section 3 presents the setup for supply chain production planning. In section 4, we introduce the Density Based Spatial Clustering of Applications with Noise (DBSCAN) model and show the empirical model performance of DBSCAN. Section 5 concludes the paper.

2 Data and Methodology

2.1 Data of Key Business Variables

In this paper, we focus on volatilities of four key business variables, namely, heating oil price, copper price, Baltic Dry Index (BDI) and exchange rate (Chinese Renminbi (RMB) against Euro). Those four types of volatilities play crucial roles in the economic activities.

The sample data we collect is for heating oil price, copper price, Baltic Dry Index (BDI) and exchange rate (RMB against Euro) from WIND database on a daily basis, where heating oil price and copper price are Chinese market prices. The sample covers the period from 1 January, 2007 to 31 December, 2018 (3,023 daily total observations for each series).

2.2 Volatility Estimation through the GARCH Model

It is noticeable that there might be heteroscedasticity effect rooted in business variable volatilities. The conditional heteroscedasticity or autocorrelation

Key Business Variables	Chi2	P-value	lag(s)
Heating Oil	2.54	0.08	1
Copper	174.27	0.00	1
BDI	490.24	0.00	1
Exchange Rate	6.68	0.01	1

Table 2 ARCH-LM test results of four key business variables.

embedded in a time series is called an autoregressive conditional heteroscedasticity (ARCH) effect. The ARCH-LM (Lagrange-Multiplier) test is used to assess the significance of ARCH effects for a particular time series. As a result, we report the ARCH-LM test results in Table 2 for the four key business variables with the null hypothesis that the time series has no ARCH effects. It is observable that all four key business variables have rejected the null hypothesis on the 10% significance level, indicating the heteroscedasticity effect existed within those time series.

On the basis, we use the generalized autoregressive conditional heteroscedasticity (GARCH) to model all volatilities in this paper, in order to show the volatility clustering features indicated by heteroscedasticity effect. Therefore, all volatilities in our paper are conditional volatilities based on GARCH model and since the time lag in ARCH-LM test is 1, we use GARCH (1, 1) model for our study.

GARCH family models are widely applied in the volatility modeling and forecasting in finance and economics, including oil market volatility (Kang et al., 2009 [13], Wei et al., 2010 [22], Yu et al., 2019 [24]); copper market volatility (Smith and Bracker, 2003 [21], Sadorsky, 2014 [20]); Baltic Dry Index (BDI) volatility (Ding et al., 2019 [7]); exchange rate volatility (Jorion, 1995 [12], Yang, 2006 [23]).

The standard GARCH (1, 1) model has the following the form:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \sigma_{t-1}^2 + \alpha_2 \varepsilon_{t-1}^2. \tag{1}$$

where σ_t is the volatility of target time series and ε_t is the residual term from the return prediction equation, which is:

$$r_t = \phi + \varepsilon_t, \tag{2}$$

where ϕ is the conditional mean, and $\varepsilon_t \sim N(0, \sigma_t^2)$.

Based on the GARCH model, we estimate conditional volatilities for all four key business variables and corresponding statistical summary is exhibited in Table 3. From Figure 1, it is clear that all volatilities are clustering into two areas, upper half area and lower half area with certain thresholds. For example, the threshold of exchange rate volatility clustering is around 0.02. The volatility level above 0.02 can be categorized as high volatility whereas the volatility level below 0.02 can be categorized as low volatility.

Key Business Variables	Obs	Mean	Max	Min
Heating Oil	3,023	0.062	0.56	0.000049
Copper	3,023	0.047	0.68	0.000057
BDI	3,023	0.75	5.42	0.00056
Exchange Rate	3,023	0.018	0.09	0.000022

Table 3 Statistical summary for conditional volatilities of four key business variables.

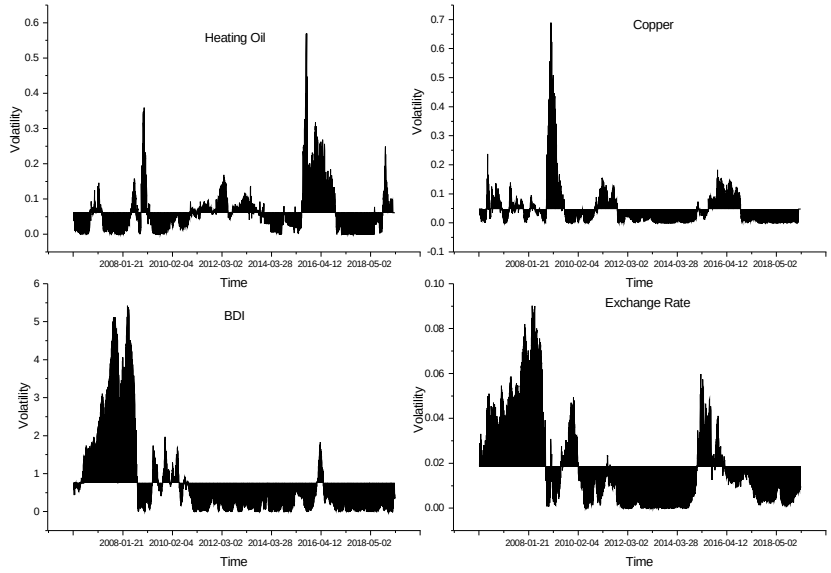


Fig. 1 Plotting of conditional volatilities for all four key business variables.

3 Production Planning Setup for Manufacturing Supply Chain

According to Maravelias and Sung (2009 [16]), supply chain for a manufacturing company mainly comprises three activities, namely, raw material procurement, manufacturing finished product from the raw material, and distributing finished products to customers. Since three stages are involved in the manufacturing process, we need to allocate time into different stages. Based on Gnoni et al. (2003 [9]), we take manufacturing production capacity around 9600 minutes per month as our example, equivalently, 20 days per month with 8 hours per day. If we presume there are 30 days in one month, then we can assume 5 days for material procurement from suppliers and 5 days for distribution to customers. Practically, purchasing of raw materials is the prerequisite of production and similarly, manufacturing products is the prerequisite of distribution. Therefore, the supply chain shall be strictly on procurement-production-distribution sequence, so we cannot interrupt this sequence for annual production scheduling.

On the basis, formulating annual production plan of supply chain is as follows:

1. We divide all days in one year into 73 blocks, with 72 blocks of 5 days plus one Remaining Block (RB). The RB can be 5 days in normal years and 6 days in leap years, which can be considered as holidays without scheduling and we leave them at the end of the year. Therefore, we only have 360 days for the annual production plan.

2. Since we have 5 days for material procurement from suppliers, 20 days for manufacturing and 5 days for distribution to customers for each month, the numbers of corresponding blocks are 12 blocks for material procurement from suppliers, 48 blocks for manufacturing and 12 blocks for distribution to customers for the annual production plan. Therefore, the number of blocks for procurement-production-distribution sequence shall be 1 block 4 blocks 1 block matched.

3. Because of the procurement-production-distribution sequence, the first five blocks in January must be one block for procurement and four blocks for production whereas the last block in December must be for distribution. As a result, those six blocks in the supply chain cannot be rescheduled.

4. Our optimized production plan of supply chain is based on the volatility clustering result, which is presented in Figure 1. The principle of optimization is that we schedule all purchase and distribution activities in the low market volatility periods to keep our cost low or keep our revenue high. The heating oil and copper prices are related to the procurement stage for the material purchase. The BDI is associated with the transportation cost and the exchange rate relates to the sale price of oversea distribution for Chinese companies. Both are related to the distribution stage. Because there are two time series related to one stage, we take heating oil price and exchange rate as main variables when there is a volatility area crash (i.e. low volatility area of one time series and high volatility area of the other time series arise simultaneously).

5. In order to formulate such optimized production plan, we adopt the decision tree method. We present an example of our decision tree method in Figure 2. As demonstrated in Figure 2, the first stage in January should be procurement and the second stage in January should be production (here the pink block stands for 5 days (i.e. 1 block), the blue production block stands for totally 20 days (i.e. 4 blocks in the figure), so there are no branches between those two blocks. Then, we have two branches after the production stage, which are procurement stage and distribution stage in pink. If the material purchase price volatility stays in the low volatility area, which can be observed in Figure 1. Then, we continue to the procurement stage, otherwise we follow the distribution branch.

6. For the upward procurement stage in pink, we can have three branches followed. We can continue to purchase materials if material purchase price volatility keeps staying in the low volatility area, or we can distribute production from the previous blue production stage if the exchange rate volatility and the transportation cost volatility stay in the low volatility area. If the

four key business variable volatilities are all in the high volatility area, we can continue the production stage alternatively.

7. By the downward distribution stage in pink, we complete one production sequence, so we have to start with the procurement stage, where only one branch is followed after the downward distribution stage. After the procurement stage, we can continue to purchase materials if material purchase price volatility keeps staying in the low volatility area. Otherwise, we continue to the production stage and because we do not finish the production stage, so we have no distribution stage followed.

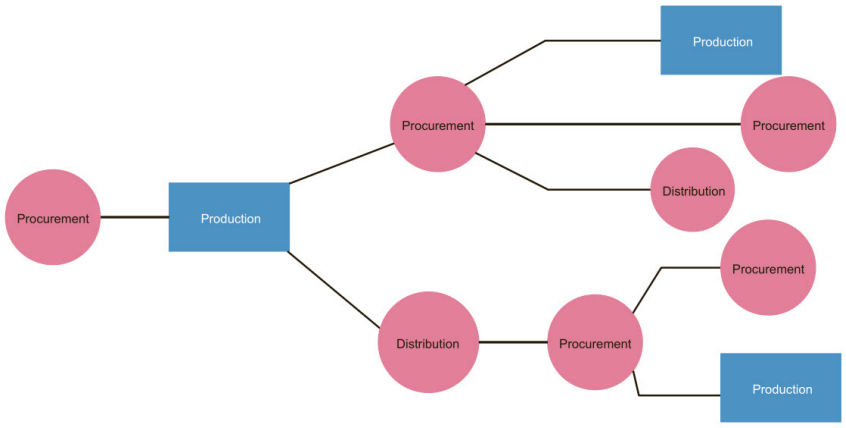


Fig. 2 Decision tree example.

4 Production Planning for Manufacturing Supply Chain based on Machine Learning Model

4.1 Density Based Spatial Clustering of Applications with Noise

In this section, we adopt an unsupervised learning machine learning algorithm, namely Density Based Spatial Clustering of Applications with Noise (DBSCAN)[8], to cluster the data for model-building. DBSCAN is a representative of density-based algorithms, which have considerable advantages over partitional and hierarchical clustering algorithms because they are not limited to finding spherical shaped based clusters but can find clusters of arbitrary shapes. Besides, density-based algorithms can effectively identify noise points, which is desired for this work because there could be a significant noise ingredient within the volatility process (Merville and Pieptea, 1989 [17]). This noise ingredient may become an interference for the information reflection role of the whole volatility process. Therefore, noise filtration of the volatility process could be helpful for our information extraction from the volatility process.

DBSCAN clusters the data points by the density. It is primarily based on the concepts of density-reachability and density-connectivity. There are two important input parameters, epsilon(eps) and minimum number of points ($minPts$). Epsilon is used to define its eps-neighborhood by using the distance around a point object. In this work, we calculate Euclidean distance between two point objects. For a given point object q, if there are at least $minPts$ point objects lie within its eps-neighborhood, then q is defined as a core point object, and all point objects within its eps-neighborhood are considered as density-reachable from q. In addition, a point object p is density-reachable if it is within the eps-neighborhood of an object that is directly density-reachable or density-reachable from q. Furthermore, two point objects p and q are density-connected if there is another point object o that both p and q are density-reachable from o.

The notions of density-reachability and density-connectivity are primarily used to find the clusters. In DBSCAN, a cluster is defined as the set of point objects that are density-connected to a particular core point object. Any point objects that are not part of a cluster will be classified as noises. This is in different with other unsupervised clustering algorithms (for example K-means), which assign every object to a cluster.

The procedure of DBSCAN is simple and it works as follows: first, all point objects are assumed to be unassigned. Next, it randomly selects a point object p, if it finds p is a core point object, it may find all the density-connected point objects based on the given eps and $minPts$. All these point objects may form a new cluster. If it finds p is not a core point object, then p is considered to be noise at the moment and it may move to next unassigned point object. Once every point object is processed, the algorithm stops.

By using DBSCAN, we attempt to cluster two set of object points. Each set contains one pair of two key business variables (heating oil price volatility against copper price volatility and BDI volatility against exchange rate volatility). It has clear advantage over volatility estimation using a single key business variable as it can capture the correlation between each two key business variables. The correlation between two time series would be highly useful because it allows us to consider volatility clustering of both time series simultaneously. The overall results are shown in Figures 3 to 6. Figure 3 presents the volatility clustering result of heating oil price volatility against copper price volatility, where the red nodes represent high volatility periods, blue nodes represent low volatility periods and green nodes represent noise within the volatility process. In order to clearly show the periods of volatility clustering, we add a time domain to make the figure 3-D, which is Figure 4. From Figure 4, we can easily identify different volatility clustering periods in the time axis. Then, we can schedule all our procurement activities into those low volatility periods. Similarly, Figure 5 presents the volatility clustering result of BDI volatility against exchange rate volatility, where the red nodes represent high volatility periods, blue nodes represent low volatility periods and green nodes represent noise within the volatility process. In order to clearly show the periods of volatility clustering, we also add a time domain to make the figure 3-D,

which is the Figure 6. From Figure 6, we can easily identify different volatility clustering periods in the time axis. Then, we can schedule all our distribution activities into those low volatility periods.

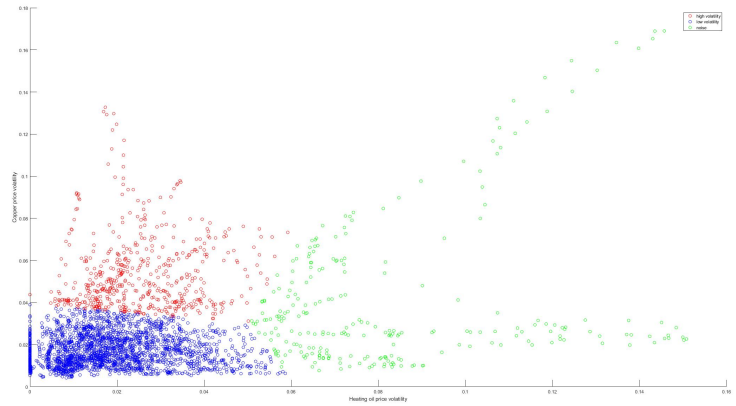


Fig. 3 Plotting of heating oil price volatility against copper price volatility.

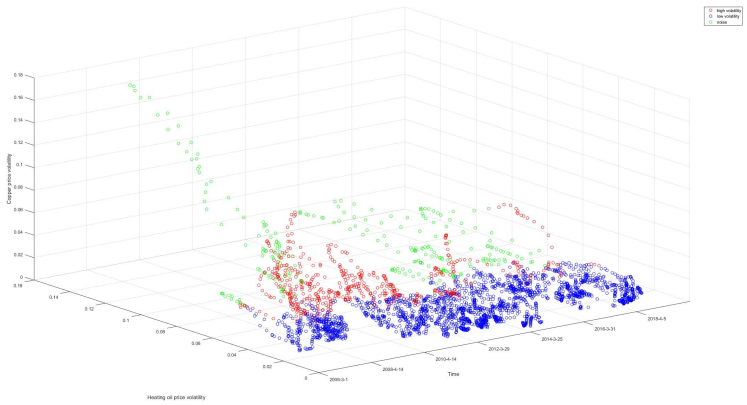


Fig. 4 Plotting of heating oil price volatility against copper price volatility with time domain.

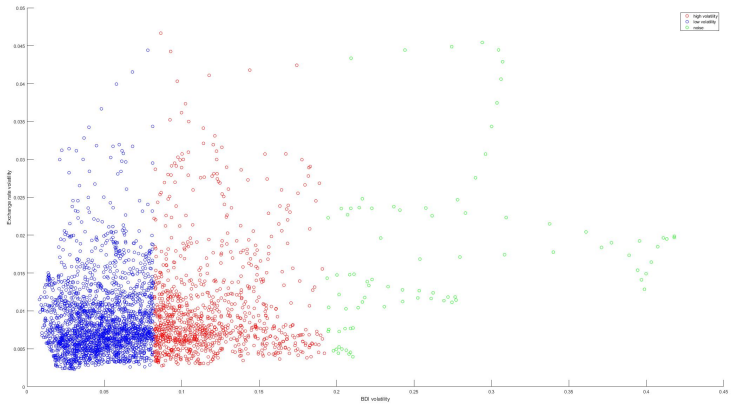


Fig. 5 Plotting of BDI volatility against exchange rate volatility.

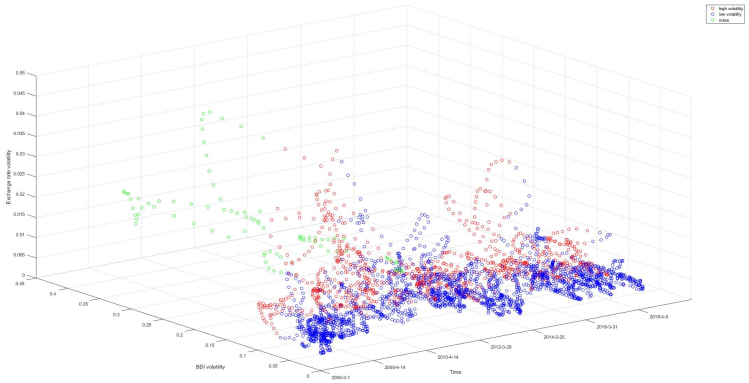


Fig. 6 Plotting of BDI volatility against exchange rate volatility with time domain.

4.2 Empirical Results for Volatility Clustering Model with Machine Learning

By using the supply chain volatility clustering scheduling technique with DBSCAN, we have developed new annual production plans for manufacturing companies. Since we take the natural log of heating oil price, copper price and BDI as mentioned in section 2.1 for the size effect elimination purpose. We now recalculate all improvement rates by taking the exponential function to bring all numbers to the original value and then we compare both models again. The novel improvement rate (NIR) will take the following form:

$$NIR_t^i = \frac{\left| e^{p_i^v} - e^{p_i^b} \right|}{e^{p_i^b}} \tag{3}$$

Key Business Variables	Overall Improvement
Heating Oil	-1.98%
Copper	-7.59%
BDI	-57.9%
Exchange Rate	1.01%
Overall	-66.46%

Table 4 Recalculation of key variable overall improvement rates by taking the exponential function for GARCH model.

Key Business Variables	Overall Improvement
Heating Oil	6.22%
Copper	9.15%
BDI	14.11%
Exchange Rate	1.51%
Overall	30.99%

Table 5 Recalculation of key variable overall improvement rates by taking the exponential function for machine learning model.

where $e^{p_i^v}$ is the original price of our model and $e^{p_i^b}$ is the original price of benchmark model, i= heating oil, copper and BDI.

We also define the overall novel improvement rate for each time series over the whole sample period:

$$OverallNIR_t^i = \frac{1}{12} \sum_{t=2007}^{2018} NIR_t^i \tag{4}$$

where i= heating oil, copper and BDI.

Table 4 reports the overall performance of our model compared with the benchmark model. Since the exchange rate has not been taken the natural log, the improvement rate for exchange rate remains the same. The overall improvement rates for other three time series, namely, heating oil price, copper price and BDI, are -1.98%, -7.59%, and -57.9% respectively. As a result, it is arguable that our GRACH model has a quite weak performance in the production planning performance. On the other hand, our machine learning model, the overall improvement rates for other three time series, namely, heating oil price, copper price and BDI, are 6.22%, 9.15%, and 14.11% respectively. As a result, it is arguable that our machine learning model can help manufacturing firms to save around 30% of the overall cost and raise about 1.5% revenue, which is more favored than the previous GARCH model.

5 Conclusion

To sum up, the main issues for this paper to accommodate is to deliver a sensible production plan based on the volatility clustering model regarding the manufacturing companies. The production plan based on volatility clustering model can be helpful in the reduction of market uncertainties. Our framework

is based on the volatility clustering model and we schedule all market related activities such as raw material purchase and product distribution into low volatility periods. Under such framework, we can thereby reduce the overall market price uncertainty faced by manufacturing companies as all market price uncertainty can be reduced through rescheduling. For the market price uncertainty, four key business variables are utilized to distil information concerning the market price uncertainty, which are heating oil price, copper price, Baltic Dry Index (BDI) and exchange rate. We firstly used the GARCH model as the volatility clustering model for the production planning, where the model performance is inferior. We argue that if we can consider volatilities in both time series simultaneously and filter the noise from the time series, the model performance could be enhanced. So we adopt a machine learning approach called Density Based Spatial Clustering of Applications with Noise (DBSCAN) to achieve these two goals and the performance actually has been improved. So we can conclude that manufacturing companies production plan can be strengthened through the market uncertainties reduction. More importantly, correlations between two time series and noise of model estimation play crucial roles in volatility clustering.

References

1. Dima Alberg, Haim Shalit, and Rami Yosef. Estimating stock market volatility using asymmetric garch models. *Applied Financial Economics*, 18(15):1201–1208, 2008.
2. Faisal Aqlan and Sarah S Lam. Supply chain optimization under risk and uncertainty: A case study for high-end server manufacturing. *Computers & Industrial Engineering*, 93:78–87, 2016.
3. David Ardia, Keven Bluteau, and Maxime Ruede. Regime changes in bitcoin garch volatility dynamics. *Finance Research Letters*, 29:266–271, 2019.
4. Xuezheng Bai, Jeffrey R Russell, and George C Tiao. Kurtosis of garch and stochastic volatility models with non-normal innovations. *Journal of Econometrics*, 114(2):349–360, 2003.
5. Martin Christopher and Matthias Holweg. Supply chain 2.0: Managing supply chains in the era of turbulence. *International Journal of Physical Distribution & Logistics Management*, 41(1):63–82, 2011.
6. Bruce Dehning, Vernon J Richardson, and Robert W Zmud. The financial performance effects of it-based supply chain management systems in manufacturing firms. *Journal of Operations Management*, 25(4):806–824, 2007.
7. Shusheng Ding, Yongmin Zhang, and Meryem Duygun. Modeling price volatility based on a genetic programming approach. *British Journal of Management*, 30(2):328–340, 2019.
8. Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Knowledge Discover and Data Mining (KDD)*, pages 226–231. AAAI Press, 1996.
9. MG Gnoni, R Iavagnilio, G Mossa, G Mummolo, and A Di Leva. Production planning of a multi-site manufacturing system by hybrid modelling: A case study from the automotive industry. *International Journal of Production Economics*, 85(2):251–262, 2003.
10. David A Guerra-Zubiaga and Robert IM Young. Design of a manufacturing knowledge model. *International Journal of Computer Integrated Manufacturing*, 21(5):526–539, 2008.

11. Kevin B Hendricks and Vinod R Singhal. Association between supply chain glitches and operating performance. *Management Science*, 51(5):695–711, 2005.
12. Philippe Jorion. Predicting volatility in the foreign exchange market. *The Journal of Finance*, 50(2):507–528, 1995.
13. Sang Hoon Kang, Sang-Mok Kang, and Seong-Min Yoon. Forecasting volatility of crude oil markets. *Energy Economics*, 31(1):119–125, 2009.
14. Jasneet Kaur, Ramneet Sidhu, Anjali Awasthi, Satyaveer Chauhan, and Suresh Goyal. A dematel based approach for investigating barriers in green supply chain management in canadian manufacturing firms. *International Journal of Production Research*, 56(1-2):312–332, 2018.
15. Shaorui Li, Vaidyanathan Jayaraman, Antony Paulraj, and Kuo-chung Shang. Proactive environmental strategies and performance: role of green supply chain processes and green product design in the chinese high-tech industry. *International Journal of Production Research*, 54(7):2136–2151, 2016.
16. Christos T Maravelias and Charles Sung. Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers and Chemical Engineering*, 33(12):1919–1930, 2009.
17. Larry J Merville and Dan R Pieptea. Stock-price volatility, mean-reverting diffusion, and noise. *Journal of Financial Economics*, 24(1):193–214, 1989.
18. Ram Narasimhan and Jayanth Jayaram. Causal linkages in supply chain management: an exploratory study of north american manufacturing firms. *Decision Sciences*, 29(3):579–605, 1998.
19. Jan Olhager and Erik Selldin. Supply chain management survey of swedish manufacturing firms. *International Journal of Production Economics*, 89(3):353–361, 2004.
20. Perry Sadorsky. Modeling volatility and correlations between emerging market stock prices and the prices of copper, oil and wheat. *Energy Economics*, 43:72–81, 2014.
21. Kenneth L Smith and Kevin Bracker. Forecasting changes in copper futures volatility with garch models using an iterated algorithm. *Review of Quantitative Finance and Accounting*, 20(3):245–265, 2003.
22. Yu Wei, Yudong Wang, and Dengshi Huang. Forecasting crude oil market volatility: Further evidence using garch-class models. *Energy Economics*, 32(6):1477–1484, 2010.
23. Lijian Yang. A semiparametric garch model for foreign exchange volatility. *Journal of Econometrics*, 130(2):365–384, 2006.
24. Lean Yu, Rui Zha, Dimitrios Stafylas, Kaijian He, and Jia Liu. Dependences and volatility spillovers between the oil and stock markets: New evidence from the copula and var-bekk-garch models. *International Review of Financial Analysis*, 2019.

MISTA 2019

The Electric Vehicle Routing Problem with Time Windows and Mobile Recharging

Ping Chen · Meng Xu · Junyuan Tan · Shu Zhang

Abstract In this paper, we present an electric vehicle routing problem with time windows and mobile recharging (EVRPTW-FMR), which aims to determine the delivery routing planning and recharging strategy simultaneously. Since the driving range of electric vehicles (EVs) is limited, EVs may need to be recharged en route. In the EVRPTW-FMR, two charging facilities can be utilized for recharging, i.e., fixed recharging stations and mobile charging vehicles. Compared to the former, mobile charging vehicles can provide much faster charging service at any designated location, however, the charging fee is higher. We define this new variant of the EVRPTW as the EVRPTW-FMR, and build a mixed integer programming formulation for this problem. We verify the model by using the CPLEX to solve small instances, and reveal the potential benefit of adopting mobile recharging. We also propose an adaptive large neighborhood search heuristic for solving large instances, and carry out numerical experiments to demonstrate its performance.

1 Introduction

Transportation activities are responsible for nearly 20% of greenhouse gas (GHG) emissions in the European Union (EU) in 2010 (EEA 2012). In order to resolve fuel consumption conflicts and reduce carbon emission, governments have been promoting

Ping Chen
Business School, Nankai University, Tianjin, China
E-mail: chenpingteda@nankai.edu.cn

Meng Xu
Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
E-mail: xm123155@163.com

Junyuan Tan
Business School, Nankai University, Tianjin, China
E-mail: tanjunyuan0217@qq.com

Shu Zhang
School of Economics and Business Administration
Chongqing University, Chongqing, China
E-mail: zhangshu@cqu.edu.cn

the development and applictions of electric vehicles (EVs). For example, the Chinese government has issued a series of policies and incentives to promote electric vehicle applications both in personal travel and logistics delivery. More and more small parcel delivery companies and electronic commerce enterprise have replaced all or part of trucks by EVs for urban logistics delivery. However, the reality that the insufficiency of charging station, the limited driving range of EVs, and long charging time are still major obstacles to EV applications.

Recently, research on electric vehicle routing problem (EVRP) has attracted a lot of attentions. In these studies, it usually assumes that EVs leave the depot with full charge, and recharge at some charging station en route before the battery runs out (Schneider et al (2014); Keskin and Çatay (2016); Pelletier et al (2016); Montoya et al (2017)). However, in practice, charging stations are sometimes inadequate, and recharging takes too long, at least 1 hour even with fast recharging. Besides, EVs would have to wait in queues, which also take time that usually could not be ignored. All these would bring challenges when customer services have strict time window constraints, and more EVs may be required. Recently, a new mode of charging service has emerged in industry, which provides recharging service to EVs by mobile charging vehicles. The charging could happen anytime at anywhere by appointment in advance. Usually, EVs get recharging at customers at the same time of loading/unloading, therefore the recharging time could be overlapped by service time of customers. In this way, there are two obvious advantages. First, EVs do not need to wait in queues; second, recharging could be finished within very short time because of the charging technology. Thus, time spent on recharging could be saved. The potential benefit is that less EVs are required because more customer could be visited in one route. However, the disadvantage is that the unit charging fee is usually higher than that of the fixed recharging stations. Therefore, the route planning and recharging strategy are worthy investigating to obtain a trade-off between the benefit and cost of the mobile recharging. Figure 1 shows an illustration example for the EVRPTW-FMR studied in this work. There are 1 depot, 3 recharging stations, and 10 customers in this figure. Three EVs are required to service all customers, where one EV does not recharge en route, one recharges at a recharging station, and the one recharge at customer C9 by mobile recharging.

To the best of our knowledge, the EVRPTW-FMR described above has not yet been stuided in literature. In this paper, we define this problem and formulate it as a mixed integer programming model, and also propose an adaptive large neighborhood search heuristic approach for solving large instances. The remainder of this paper is organized as follows. Literature is reviewed in Section 2. In Section 3, we define the EVRPTW-FMR, and present a mathematical formulation. Section 4 presents the details of the proposed heuristic approach. Computational results and analysis are given in Section 5, and Section 6 concludes this paper.

2 Literature Review

The EVRPTW-FMR can be seen as an extension of the EVRPs, thus we briefly review the related work on EVRPs in this section. Pelletier et al (2016) give a recent review of EVRPs in goods distribution, so readers are referred to their work for more information.

Gonçalves et al (2011) investigate a VRP with pickup and delivery with a mixed fleet including both conventional vehicles and EVs. They suppose each EV has a fixed charging time and apply their model to a real case of a Portuguese battery distrib-

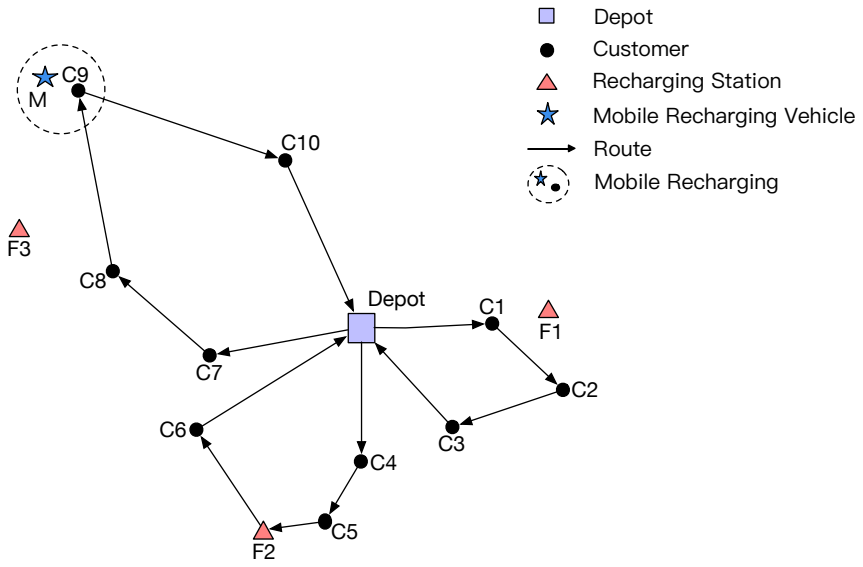


Fig. 1 An Illustration Example of EVRPTW-FMR

utor. Erdoğan and Miller-Hooks (2012) formulate the green vehicle routing problem (GVRP), in which recharge stations are considered and the charging time is a constant. Schneider et al (2014) first formally introduce the electric vehicle routing problem with time windows and recharge stations(E-VRPTW) by extending their work. In the E-VRPTW, the EVs is fully recharged and the recharging time depends on the battery status when the EV arrives at the recharging station. A hybrid heuristic that combines a variable neighborhood search algorithm with a tabu search heuristic is developed for solving the E-VRPTW. Afroditi et al (2014) develop a model for the EVRP with the constraints of capacity, time windows, etc. Keskin and Çatay (2016) propose an adaptive large neighborhood search algorithm for the EVRPTW, and Desaulniers et al (2016) presents exact branch-price-and-cut algorithms. Montoya et al (2017) introduces the electric vehicle routing problem with non-linear charging functions and extends the modified multi-space sampling heuristic to solve this problem.

The electric fleet size and mix vehicle routing problem (E-FSMVRP) is the extension variant of the EVRP. Goeke and Schneider (2015) propose the electric vehicle routing problem with time windows and mixed fleet (E-VRPTWMF) to optimize the routing of a mixed fleet of electric commercial vehicles and conventional internal combustion commercial vehicles, and develop an adaptive large neighborhood search algorithm that is enhanced by a local search for intensification. Hiermann et al (2016) introduce the electric fleet size and mix vehicle routing problem with time windows and recharging stations (E-FSMFTW) and develop a hybrid heuristic to solve it, which combines an adaptive large neighborhood search with an embedded local search and labeling procedure for intensification. Penna et al (2016) propose an iterative local search algorithm coupled with a set partitioning model to solve E-FSMTW. Desaulniers et al (2016)

present exact algorithms for electric vehicle routing problems with time windows, and conclude that allowing multiple and partial recharges on one route contributes to both costs and the number of vehicles required reduction, compared with other scenarios. Yang and Sun (2015) introduce the electric vehicles battery swap stations.

Some studies extend the EVRP research by introducing partial recharging. Felipe et al (2014) extend the model by Erdoğan and Miller-Hooks (2012) by allowing partial recharges and taking multiple charging technologies. Bruglieri et al (2015) propose a problem to minimize the total time as well as the number of the employed EVs, while partial recharging is allowed. In their study, a variable neighborhood search branching (VNSB) is designed to solve the problem. Sassi et al (2015) consider the heterogenous electric vehicle routing problem with time dependent charging costs and a mixed fleet (HEVRP-TDMF). Ding et al (2015) investigate an E-VRPTW with pickup and delivery and partial charging. Desaulniers et al (2016) consider four charing strategies: a single charge or multiple charges per route and fully recharge only, multiple recharges per route and batteries are fully charged, at most a single recharge per route and partial recharges, and multiple partial recharges. Schiffer and Walther (2017) propose the electric location routing problem with time windows, which simultaneously determines the routing and charging stations locations. Paz et al (2018) introduce the multi-depot electric vehicle location-routing problem with time windows.

Compared to the work on EVRPs recharging at stations, few studies are on EVRPs recharging at customer locations. To the best of our knowledge, only Conrad and Figliozzi (2011) has investigated a similar problem. The authors presented a recharging vehicle routing problem (RVRP) that allows EVs recharging at some customer vertices simultaneously with servicing customers. The aim is primarily to minimize the number of vehicles needed, and secondarily to minimize the total cost. Compared to their study, there are three major differences between the EVRPTW-FMR and their problem. Firstly, recharging could happen at any customer location in the EVRPTW-FMR, instead of some specific customer locations in Conrad and Figliozzi (2011); secondly, both fixed recharging stations and mobile recharging vehicles can be utilized in the EVRPTW-FMR, which indicates that the selection of facility also needs to be determined; thirdly, it is required to full recharging in Conrad and Figliozzi (2011) while it is not necessary in the EVRPTW-FMR, thus the charging time is also a decision variable. Because of these differences, the EVRPTW-FMR is more complicated to be solved.

3 Problem Definition and Formulation

Based on the definition of the E-VRPTW in Schneider et al (2014), we define the EVRPTW-FMR as a directed graph $G = (V_{0,N+1}, A)$. Here, let $V_{0,N+1} = \{0\} \cup V \cup \{N+1\}$, where V denotes the union of customer set I, F_1' the set of dummy vertices to allow multiple visits to each recharging station, and F_2 the set of potential recharging locations by mobile vehicles. In this work, F_2 is essentially the same as I. Each customer in I is associated with a demand u_i, a service time s_i, and a time windows $[e_i, l_i]$. $\{0\}$ and $\{N+1\}$ represent the start and return depots, respectively. The set of arcs A is defined as $\{(i,j)|i,j \in V'_{0,N+1}, i \neq i\}$. For each arc $(i,j) \in A$, it is associated with a travel time t_{ij} and a distance d_{ij}. It is required that a fleet of homogeneous EVs leave the depot full charged, visit a sequence of customers, and finally return to the depot. The battery capacity is denoted as Q. The electricity consumption rate is denoted as

Table 1 Mathematical notations

Problem	
$0, N+1$	depot vertices;
F_1'	set of recharging stations;
F_1'	set of dummy vertices of recharging stations;
F_2	set of vertices being charged by mobile recharging vehicles;
F	set of recharging visits, $F = F_1' + F_2$;
F_0	set of recharging visits including depot;
I	set of customers, $I = (1, ..., N)$;
I_0	set of customers including depot 0;
V_0	set of customers and recharging visits including depot 0;
V_{N+1}	set of customers and recharging visits including depot $N+1$;
$V_{0,N+1}$	set of customers and recharging visits including depot 0and $N+1$;
EVs parameters	
Q	battery capacity;
C	load capacity;
v_0	running speed of the EVs;
h	charge consumption rate;
c_t	travel cost per unit distance;
c_v	fixed cost per EV used;
M	a very large constant;
Recharging parameters	
g_f	charging rate of recharging stations;
c_f	charging cost of recharging stations;
g_m	charging rate of mobile charging vehicles, $g_f < g_m$;
c_m	charging cost of mobile charging vehicles, $c_f < c_m$;
Customers characteristics	
q_i	demand of customer i, $i \in I$;
p_i	service time of customer i, $i \in I$;
$[e_i, l_i]$	time window of customer i, $i \in I$;
d_{ij}	distance between vertices i and j, $i, j \in V_{0,N+1}$;
t_{ij}	travel time between vertices i and j, $i, j \in V_{0,N+1}$;
Decision variables	
x_{ij}	equals to 1 if arc (i, j) is traveled, $i, j \in V_{0,N+1}$;
τ_i	the time of arrival at vertex i, $i \in V_{0,N+1}$;
u_i	the remaining cargo when EV arrivals at vertex i, $i \in V_{0,N+1}$;
Y_i	the remaining battery capacity when EV leaves vertex i;
y_i	the remaining battery capacity when EV arrives at vertex i.

h, i.e., the electricity consumed per unit distance. EVs could be recharged at either a recharging station or at some customer by a mobile recharging vehicle. However, the recharging rate and cost are different. The recharging rate is denoted as g_f and g_m, respectively; the recharging cost per unit is represented by p_f and p_m (suppose $p_m > p_f$), respectively. The number and locations of available fixed recharging station are known in advance, while the number of mobile recharging facilities is unlimited with the practical consideration. The aim is to determine the route planning and recharging scheduling, while minimize the total cost, which includes the usage cost for EVs, the total travel cost, and the total recharging cost.

We formulate the EVRPTW-FMR as a mixed integer programming model based on the E-VRPTW model in Schneider et al (2014).

$$\min \sum_{j \in V_{N+1}} c_v x_{0j} + \sum_{i \in V_0, j \in V_{N+1}, i \neq j} c_t d_{ij} x_{ij} + c_f \sum_{i \in F_1} (Y_i - y_i) + c_m \sum_{i \in F_2} (Y_i - y_i) \quad (1)$$

s.t.

$$\sum_{j \in V_{N+1}, i \neq j} x_{ij} = 1 \quad \forall i \in I \tag{2}$$

$$\sum_{i \in V_0, i \neq j} x_{ij} - \sum_{i \in V_{N+1}, i \neq j} x_{ji} = 0 \quad \forall j \in V \tag{3}$$

$$\sum_{j \in V_{N+1}, i \neq j} x_{ij} \leqslant 1 \quad \forall i \in F \tag{4}$$

$$\sum_{j \in F, i \neq j} x_{ij} = 0 \quad \forall i \in F \tag{5}$$

$$\tau_i + (t_{ij} + p_i)x_{ij} - M(1 - x_{ij}) \leqslant \tau_j \quad \forall i \in I_0, \forall j \in V_{N+1}, i \neq j \tag{6}$$

$$\tau_i + t_{ij}x_{ij} + g_f(Y_i - y_i) - (M + g_f Q)(1 - x_{ij}) \leqslant \tau_j \quad \forall i \in F_1, \forall j \in V_{N+1}, i \neq j \tag{7}$$

$$\tau_i + t_{ij}x_{ij} + \max[0, g_m(Y_i - y_i) - p_i] - (M + g_m Q)(1 - x_{ij}) \leqslant \tau_j \quad \forall i \in F_2, \forall j \in V_{N+1}, i \neq j \tag{8}$$

$$e_j \leqslant \tau_j \leqslant l_j \quad \forall j \in V_{0,N+1} \tag{9}$$

$$0 \leqslant u_j \leqslant u_i - q_i x_{ij} + C(1 - x_{ij}) \quad \forall i \in V_0, j \in V_{N+1}, i \neq j \tag{10}$$

$$0 \leqslant u_0 \leqslant C \tag{11}$$

$$0 \leqslant y_j \leqslant y_i - (hd_{ij})x_{ij} + Q(1 - x_{ij}) \quad \forall i \in I, j \in V_{N+1}, i \neq j \tag{12}$$

$$0 \leqslant y_j \leqslant Y_i - (hd_{ij})x_{ij} + Q(1 - x_{ij}) \quad \forall i \in F_0, j \in V_{N+1}, i \neq j \tag{13}$$

$$y_i \leqslant Y_i \leqslant Q \quad \forall i \in F_0 \tag{14}$$

$$\sum_{i \in I} d_{ij} = \sum_{i \in I} d_{ji} = 0 \quad \forall j \in F_2 \tag{15}$$

$$\sum_{i \in I} t_{ij} = \sum_{i \in I} t_{ji} = 0 \quad \forall j \in F_2 \tag{16}$$

$$p_i - p_j = 0 \quad \forall i \in I, \forall j \in F_2 \tag{17}$$

$$t_{ij} = \frac{d_{ij}}{v_0} \quad \forall i \in V_0, \forall j \in V_{N+1}, i \neq j \tag{18}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in V_0, \forall j \in V_{N+1}, i \neq j \tag{19}$$

Equation (1) is the objective that aims to minimize the total cost, i.e., the sum of usage cost of EVs, the total travel cost, and the recharging cost at recharging stations as well as that by mobile charging vehicles, respectively. Constraint (2) ensures that each customer is visited by one EV exactly once; (3) enforces the connectivity of visits to recharging facilities. Constraints (4) and (5) handle the characteristics of visits to the recharging facilities. Constraints (6)-(8) guarantee time windows feasibility for arriving at customers and depots, where (7) and (8) guarantee the time feasibility for EVs leaving fixed recharging stations and mobile vehicles, respectively. Constraint (9) ensures that time window constraint. Constraints (10) and (11) enforce that each vertex is fulfilled its demand. Constraints (12)-(14) guarantee that the battery charge is always above 0. Constraints (15)-(17) ensure that the EVs can be recharged by mobile recharge vehicles at any customer location. Constraint (18) denotes the relationship between travel distance and travel time. Finally, constraint (19) defines the range of decision variable x_{ij}.

4 A Heuristic Approach

We propose an adaptive variable neighborhood search heuristic (ALNS) for solving large scale instances in real-life applications. The ALNS was firstly developed by Stefan Ropke (2006), which a strongly performed heuristic for solving combinatorial optimization problems, especially in VRP variants (Stefan Ropke (2006); Dayarian et al (2016); Li et al (2016); Ghilas et al (2016); Yossiri et al (2014); Renaud et al (2013); Maximilian and Grit (2018)). It extends the large neighborhood search by Shaw (1997) by using several large neighborhoods in an adaptive way. Algorithm 1 shows the framework of this heuristic. Firstly, an initial solution to the EVRPTW-FMR is generated; next, a group of ruin and recreate procedures taken from literature as well as designed specific to the EVRPTW-FMR are described, which are applied for solution improvement. Finally, the weight adjustment strategy of each removal-insertion pair and the solution acceptance criteria are presented. More details would be presented in the following sections.

Algorithm 1 The Framework of the ALNS

Input: Generate an initial solution s_0; Define removal operators N^- and insertion operators N^+;
Output:
Set the best solution found so far $s_{best} = s_0$;
while Stopping criteria are not met **do**
  Let $s' = s_0$;
  Select operators: select a removal operator $removal \in N^-$, and an insertion operator $insertion \in N^+$;
  Apply Removal: $s'_{partial} = r(s', p)$, i.e., p vertices are to be removed from solution s', and kept in a list L;
  Apply Insertion: $s'' = i(s'_{partial}, p)$, i.e., insert vertices from L to $s'_{partial}$ and generate a new solution s'';
  if $f(s'') \leq f(s_{best})$ **then**
    $s_{best} = s''$;
  end if
  Set the candidate solution s to s' by **Acceptance Criterion**.
end while
return s_{best}.

4.1 Computation of recharging time

The charging time is an essential issue to solution quality. In our heuristic, we define different "target energy" for charging at recharging stations and at customers by mobile charging vehicles, respectively. If recharging is provided by a mobile charging vehicle, the recharging time $chargedT$ at customer i is computed as Equation (20), where arc (i,j) belongs to the route, and j_{bs} denotes the nearest recharging station from node j.

$$chargedT = min\{s_i, t_{ij} + t_{j,j_{bs}}\} * g_m \tag{20}$$

In another case, i.e. charging at a recharging station, the basic principle is to charge as much as possible. To be specific, the recharging time is determined by Equation (21), where $remainQ$ denotes the remaining energy when the EV arrives at the station. Here, the concept of $maxT$ is defined as the maximum free time available for charging, provided that the time windows for subsequent nodes are guaranteed.

$$chargedT = min\{maxT, (Q - remainQ)/(h * v\} \tag{21}$$

4.2 Initial Solution Generation

We generate an initial solution by a two-phase procedure. In the first phase, several routes without any recharging facilities are generated according to the Solomon I1 insertion heuristic Solomon (1987). To be specific, select the unrouted customer which is farthest from the depot as the seed for a new route; insert other customers into the best position based on the I1 rule until no more customer can be inserted; then, generate a new route and repeat this procedure until all customers are routed.

In the first phase, both the capacity and time windows constraints must be satisfied for each route, while the constraint of EVs electricity is temporarily ignored. In the second phase, recharging facilities are inserted to guarantee the EVs electricity capacity constraint to be satisfied. In order to generate an initial solution with both fixed recharging stations and mobile charging vehicles, we use the same procedure as

Recharging Facilities Insertion Operators which is discussed below to insert recharging facilities. It is worth mentioning that the the final complete solution is not necessarily feasible because of the recharging strategy.

4.3 Removal Procedures

4.3.1 Customer removal operators

Shaw removal: In this procedure, similar customers are removed according to the relatedness, which is defined a $R_{ij} = \phi_1 d_{ij} + \phi_2 |e_i - e_j| + \phi_3 l_{ij} + \phi_4 |D_i - D_j|$. ϕ_1, ϕ_2, ϕ_3 and ϕ_4 are four *Shaw* parameters which respectively decide the weights related to distance, time, route and demand in the calculation of two customers' relatedness. l_{ij} equals 1 if the two customers are in different routes, otherwise equals -1. The smallerR_{ij} is, the more similar the two customers are. In this procedure, firstly, a random customer is removed. Then it sorts all the value of R_{ij} in the non-descending order and removes $numtoRelax$ customers with top $numtoRelax$ smallest relatedness value from the routes.

Other three derived version of the *Shaw* removal operators, named *demand*, *distance*, and *time*-based removals, can be realized by setting the parameters ϕ_4, ϕ_1, and ϕ_2 to be 1, respectively, while maintaining other parameters to be 0.

Worst Distance Removal: In this procedure, we remove $numtoRelax$ customers with longest distances which is the sum of distances of the precursor and the successor customers in the route.

Random Route Removal: The random route removal pick one route randomly and removes all the customers in this route, which aims to reduce the number of routes.

4.3.2 Recharging Removal Operators

We simply use the *random* removal and *worst* removal operators for recharging station removal. In the first procedure, we randomly select N_{rs} stations and remove them. In the second one, we randomly remove a recharging station, and then remove another one that is farthest from this one until N_{rs} stations have been removed. For the mobile charging vehicles, we just randomly remove N_{rv} recharging vehicles from the solution.

4.4 Insertion Procedures

4.4.1 Customer Insertion Operators

We apply the *greedy* and *regret* rules in customer insertion. In the *Greedy Insertion*, for each customer i in the removal list, this procedure determine the best one for i that increases the cost least from all feasible positions. The *Regret Insertion* is used to prevent the myopic property of the greedy insertion. The basic idea is to calculate the difference between the cost of the first and *k-th* best insertions of the customers and insert the one with the highest difference to its best position.

4.4.2 Recharging Insertion Operators

The rules of insertions of recharging facilities are similar to those in customer insertions. For the first customer i at which the EV electricity status is negative, try to insert a station between its previous customer $i-1$ and i. If it is an feasible position, which indicates that the remaining electricity can support the EV to drive to the station and the time window constraints are also not violated, insert the nearest station at this position, and the recharging the electricity as much as possible while ensure that the EV arrives to i before l_i; otherwise, continue to check the positions backward until find a feasible position. It should be noted that during this procedure, the situation that would result in two consecutive recharging stations should be avoided, considering the practice rationality and the EVRPTW-FMR formulation. Update the EV electricity status forward. If there is still customers at which the EV electricity is negative, repeat this procedure.

5 Numerical Results

In this section, we present the instances and numerical results to validate the model and the performance of the proposed ALNS heuristic. Firstly, we show the instances and parameters setting in our experiments. Then, solutions to the small instances by the commercial solver CPLEX are given. Third, we evaluate the performance of the proposed ALNS by comparing its resutls on small instances with those by the CPLEX, and also present the results on large scale instances. Finally, we analyze the potential benefit from the use of mobile recharging vehicles. The algorithm was implemented in Java, and runs on Intel(R) Core(Tm) i7-7700 CPU, 3.60 GHz. We use CPLEX V12.8 to solve the small instances.

5.1 Test Instances and Parameters Setting

To validate the model and evaluate the ALNS algorithm proposed in this paper, we conducted experiments on both small and large scale instances. The basic data, like information related to recharging stations and customers are taken from the E-VRPTW benchmark instances by Schneider et al (2014). Other additional parameters are set by referring the survey of real applications. The E-VRPTW benchmark instances include 36 small scale instances, which are divided into three categories with 5, 10, and 15 customers, respectively. Besided, 56 large scale instances with 100 customers are adopted from Solomon (1987) instances, which are divided into six categories by customers distribution, i.e., clustered set (C), randomly distributed set (R), and a combination of both (RC). Schneider et al (2014) added recharging stations to each instances. Also, they set the parameters including the vehicle fuel tank capacity Q, the fuel consumption rate r, the inverse refueling rate g, and the average velocity v.

We set v and r the same as those in E-VRPTW. As for other parameters, we set the values as follows. Considering the ratios of cost and recharging rate between fixed recharge stations and mobile recharge vehicles, we let g_f equal to original data of g in Schneider et al (2014) instances, and let g_m to be three times of gf , respectively. As the cost coefficient for recharging, c_f and c_m are set to be 1.8 and 5 , respectively. Finally, the fixed cost per vehicle c_t is supposed to be 1000; the cost coefficient for travel

distance c_v equals to 1. Besides, it is worth mentioning that, we set the energy capacity Q as maximum of the data in Schneider et al (2014) and the maximum distance in the node network, to avoid two recharging station visits appear in a route consecutively, because this is unreasonable and unlikely acceptable in practice.

5.2 Solutions to Small Instances by CPLEX

We use 36 small-sized instances with 5, 10, and 15 customers to validate the model of the EVRPTW-FSR. Table 2 shows the results in details. The column *cost* denotes the total cost; the column f denotes the total travel distance; the columns rf and rm denote the total electricity charging at recharging stations, and by mobile recharging vehicles, respectively. The last column *time* denotes the computing time in term of seconds. We set the upper computing time as 7200 seconds.

From Table 2, it can be seen that all 12 instances with 5 customers only are solved optimally within 10 seconds. There are 10 instances with 10 customers being solved optimally while 2 instances not. For the rest 12 instances with 15 customers, most instances can not be solved optimally within 7200 seconds, but instances *C106C15*, *C208C15*, and *R105C15* are three exceptions. These results validate the effectiveness of the model for the EVRPTW-FMR built in this paper. In addition, from Table 2, we can also observe that there are 18 out of 36 instances which indeed take advantage of the mobile recharging.

To more accurately evaluate the benefit of using mobile recharging, we run the CPLEX again by setting the parameter c_m, i.e., the unit charging cost by mobile charging vehicles, to be a quite large number. Thus it can be ensured that no mobile recharging is used during the routes. We present these results in Table 3, in these columns denoted as *EVRPTW-FR*. We also make a comparison with the results of *EVRPTW-FMR* in terms of total cost (*cost*), the number of vehicles (m), and the total travel distance (f). Column *Saving*(%) shows the savings in the objective *cost*. From this table, it can be observed that 18 out of 36 instances can benefit from the use of mobile charging vehicles, where the savings of 8 instances are even more than 10%. These savings are mainly attribute to the reduction of the number of EVs required.

5.3 ALNS Results

In this section, we present the numerical results by the proposed ALNS. Firstly, we present the obtained results on small instances with 5, 10, and 15 customers in Table 4, and compare these results with those obtained by CPLEX. Then, results on large scale instances with 100 customers are shown in Table 5. Table 4 shows that the ALNS performs well on most small instances, while for some cases the performance is not so satisfactory and needs to be further investigated. Due to time consideration, the *max_iteration* is set to be 20. Results in Table 5 demonstrate that the heuristic could solve large scale instances effectively. More improvements on solution quality could be expected with a bigger *max_iteration* setting.

Table 2 Results of small instances by CPLEX

Instances	cost	m	f	rf	rm	time
C101C5	2364.40	2	245.42	87.98	1.94	0.58
C103C5	1304.50	1	168.16	88.15	2.26	1.00
C206C5	1490.69	1	242.56	164.81	0.00	0.48
C208C5	1287.34	1	164.34	86.59	0.00	0.19
R104C5	1456.74	1	143.21	22.10	60.48	0.37
R105C5	2185.19	2	153.01	29.99	1.76	0.13
R202C5	1225.95	1	128.88	68.25	0.00	3.08
R203C5	1327.13	1	179.00	102.18	0.00	9.33
RC105C5	2327.89	2	233.77	78.27	0.00	0.84
RC108C5	2353.12	2	249.94	80.22	1.75	2.00
RC204C5	1330.21	1	176.12	94.30	3.91	9.90
RC208C5	1296.81	1	167.98	90.23	0.00	5.60
C101C10	3563.93	3	380.26	144.29	0.00	25.63
C104C10	1637.79	1	200.62	38.77	81.50	7207.51
C202C10	1506.80	1	248.75	171.00	0.00	226.87
C205C10	1418.87	1	218.13	134.05	0.62	3.38
R102C10	3211.33	3	219.85	19.69	0.00	40.85
R103C10	2248.06	2	176.65	53.19	2.20	5687.69
R201C10	1518.78	1	241.51	180.88	0.00	109.69
R203C10	1449.48	1	218.65	152.53	0.00	7208.36
RC102C10	3495.04	3	379.18	100.49	2.16	5.62
RC108C10	2448.89	2	283.12	76.12	17.07	3729.76
RC201C10	1873.74	1	378.08	290.74	9.59	63.72
RC205C10	2534.32	2	325.98	151.96	0.00	2930.44
C103C15	2658.03	2	348.84	183.67	9.67	7208.46
C106C15	2478.58	2	270.73	98.17	17.06	1064.50
C202C15	2663.83	2	376.44	201.49	0.00	7204.92
C208C15	1608.97	1	287.75	201.59	3.18	1528.07
R102C15	5328.10	5	338.16	31.98	0.00	7208.29
R105C15	3502.38	3	317.09	129.75	3.03	2414.59
R202C15	1698.08	1	310.80	235.55	5.09	7208.10
R209C15	1609.97	1	278.93	214.90	0.00	7207.65
RC103C15	3329.50	3	313.47	43.73	0.00	7206.78
RC108C15	3354.54	3	318.20	55.55	0.00	7209.83
RC202C15	2295.61	1	534.87	434.80	17.01	7242.87
RC204C15	1624.54	1	297.87	214.58	0.00	7213.95

6 Conclusions and Future Work

In this work, we study the EVRPTW-FMR, which is an extension of the EVRPTW that takes advantage of the mobile recharging. Besides the routing planning, this problem requires to decide which recharging mode to use, when and where to recharge, and how much to recharge, which brings a challenges to problem solving. We develop a mathematical model of the EVRPTW-FMR, and solve small instance optimally by CPLEX. We design different charging time determination strategy for different type of charging mode, and propose an adaptive large neighborhood search approach to solve large instances. Numerical results validate the model and demonstrate the effectiveness of the proposed heuristic algorithm. Besides, results indicate that taking advantage of mobile recharging can bring potential cost reduction by reducing the number of vehicles required. Next, we will continue work on the heuristic algorithm performance improvement, parameter analysis, and constraint influence investigation,.

Table 3 Benefit of taking advantage of mobile charging vehicles

Instances	EVRPTW-FR			EVRPTW-FMR			Saving(%)
	cost	m	f	cost	m	f	
C101C5	2390.24	2	257.75	2364.40	2	245.42	1.09
C103C5	1316.01	1	175.37	1304.50	1	168.16	0.88
C206C5	1490.69	1	242.56	1490.69	1	242.56	0.00
C208C5	1287.34	1	164.34	1287.34	1	164.34	0.00
R104C5	2144.17	2	136.69	1456.74	*1*	143.21	47.19
R105C5	2203.69	2	156.08	2185.19	2	153.01	0.85
R202C5	1225.95	1	128.88	1225.95	1	128.88	0.00
R203C5	1327.13	1	179.00	1327.13	1	179.00	0.00
RC105C5	2327.89	2	233.77	2327.89	2	233.77	0.00
RC108C5	2357.88	2	253.93	2353.12	2	249.94	0.20
RC204C5	1341.19	1	185.16	1330.21	1	176.12	0.83
RC208C5	1296.81	1	167.98	1296.81	1	167.98	0.00
C101C10	3563.93	3	380.26	3563.93	3	380.26	0.00
C104C10	2297.46	2	225.66	1637.79	*1*	200.62	40.28
C202C10	1506.80	1	248.75	1506.80	*1*	248.75	0.00
C205C10	1581.70	1	281.51	1418.87	1	218.13	11.48
R102C10	3211.33	3	219.85	3211.33	3	219.85	0.00
R103C10	2248.28	2	179.44	2248.06	2	176.65	0.01
R201C10	1518.78	1	241.51	1518.78	1	241.51	0.00
R203C10	1449.48	1	218.65	1449.48	1	218.65	0.00
RC102C10	3506.56	3	384.06	3495.04	3	379.18	0.33
RC108C10	3277.31	3	296.31	2448.89	*2*	283.12	33.83
RC201C10	1896.56	1	398.66	1873.74	1	378.08	1.22
RC205C10	2534.32	2	325.98	2534.32	2	325.98	0.00
C103C15	3379.26	3	307.35	2658.03	*2*	348.84	27.13
C106C15	3343.56	3	280.31	2478.58	*2*	270.73	34.90
C202C15	2666.02	2	377.28	2663.83	2	376.44	0.08
C208C15	2369.63	2	257.05	1608.97	*1*	287.75	47.28
R102C15	5328.10	5	338.16	5328.10	5	338.16	0.00
R105C15	3514.75	3	325.58	3502.38	3	317.09	0.35
R202C15	1788.08	1	351.68	1698.08	1	310.80	5.30
R209C15	1692.69	1	310.74	1609.97	1	278.93	5.14
RC103C15	3329.50	3	313.47	3329.50	3	313.47	0.00
RC108C15	3354.54	3	318.20	3354.54	3	318.20	0.00
RC202C15	2597.10	2	344.64	2295.61	*1*	534.87	13.13
RC204C15	1624.54	1	297.87	1624.54	1	297.87	0.00

Acknowledgements This work was supported by the National Natural Science Foundation of China [grant numbers 71701107, 71601024, 71971032].

References

Afroditi A, Boile M, Theofanis S, Sdoukopoulos E, Margaritis D (2014) Electric vehicle routing problem with industry constraints: Trends and insights for future research. *Transportation Research Procedia* 3:452–459

Bruglieri M, Pezzella F, Pisacane O, Suraci S (2015) A variable neighborhood search branching for the electric vehicle routing problem with time windows. *Electronic Notes in Discrete Mathematics* 47:221–228

Table 4 Results by ALNS on 36 small instances

Instances	CPLEX				ALNS				Gap
	cost	m	f	time	cost	m	f	time	
C101C5	2364.40	2	245.42	0.58	2364.40	2	245.42	0.23	0.00
C103C5	1304.50	1	168.16	1.00	1316.01	1	175.37	0.22	0.88
C206C5	1490.69	1	242.56	0.48	1490.69	1	242.56	0.15	0.00
C208C5	1287.34	1	164.34	0.19	1287.34	1	164.34	0.12	0.00
R104C5	1456.74	1	143.21	0.37	1511.71	1	132.81	0.19	3.77
R105C5	2185.19	2	153.01	0.13	2203.69	2	156.08	0.14	0.85
R202C5	1225.95	1	128.88	3.08	1262.99	1	143.13	0.11	3.02
R203C5	1327.13	1	179.00	9.33	1327.13	1	179.00	0.16	0.00
RC105C5	2327.89	2	233.77	0.84	2347.47	2	241.30	0.11	0.84
RC108C5	2353.12	2	249.94	2.00	2357.88	2	253.93	0.09	0.20
RC204C5	1330.21	1	176.12	9.90	1341.19	1	185.16	0.10	0.83
RC208C5	1296.81	1	167.98	5.60	1296.81	1	167.98	0.13	0.00
C101C10	3563.93	3	380.26	25.63	3612.37	3	393.55	0.98	1.36
C104C10	1637.79	1	200.62	7207.51	1990.75	1	234.22	1.86	21.55
C202C10	1506.80	1	248.75	226.87	1749.90	1	304.21	1.66	16.13
C205C10	1418.87	1	218.13	3.38	1584.56	1	282.61	1.98	11.68
R102C10	3211.33	3	219.85	40.85	3317.45	3	249.19	1.31	3.30
R103C10	2248.06	2	176.65	5687.69	2321.08	2	207.44	2.23	3.25
R201C10	1518.78	1	241.51	109.69	1552.83	1	254.60	2.14	2.24
R203C10	1449.48	1	218.65	7208.36	1448.06	1	218.10	1.51	-0.10
RC102C10	3495.04	3	379.18	5.62	3678.09	3	395.16	1.30	5.24
RC108C10	2448.89	2	283.12	3729.76	2922.85	2	329.89	1.16	19.35
RC201C10	1873.74	1	378.08	63.72	1933.49	1	412.86	2.35	3.19
RC205C10	2534.32	2	325.98	2930.44	2577.18	2	342.46	1.71	1.69
C103C15	2658.03	2	348.84	7208.46	2958.00	2	345.56	10.55	11.29
C106C15	2478.58	2	270.73	1064.50	2634.49	2	284.41	10.00	6.29
C202C15	2663.83	2	376.44	7204.92	2751.09	2	410.00	11.52	3.28
C208C15	1608.97	1	287.75	1528.07	1929.60	1	343.52	10.21	19.93
R102C15	5328.10	5	338.16	7208.29	5493.52	5	420.83	4.63	3.10
R105C15	3502.38	3	317.09	2414.59	3628.94	3	369.50	8.95	3.61
R202C15	1698.08	1	310.80	7208.10	2324.59	1	536.19	10.40	36.89
R209C15	1609.97	1	278.93	7207.65	1704.61	1	315.33	14.28	5.88
RC103C15	3329.50	3	313.47	7206.78	4075.16	3	415.11	6.80	22.40
RC108C15	3354.54	3	318.20	7209.83	3474.91	3	361.24	6.84	3.59
RC202C15	2295.61	1	534.87	7242.87	2720.76	2	392.20	12.45	18.52
RC204C15	1624.54	1	297.87	7213.95	1851.56	1	385.18	15.56	13.97

Conrad RG, Figliozzi MA (2011) The recharging vehicle routing problem. In: Doolen T, Aken EV (eds) Proceedings of the 2011 Industrial Engineering Research Conference, Reno, NY

Dayarian I, Crainic TG, Gendreau M, Rei W (2016) An adaptive large-neighborhood search heuristic for a multiperiod vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review* 95:95–123

Desaulniers G, Errico F, Irnich S, Schneider M (2016) Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research* 64:1388–1405

Ding N, Battay R, Kwon C (2015) Conflict-free electric vehicle routing problem with capacitated charging stations and partial recharge. *https://wwwchkwon net/papers*

Erdoğan S, Miller-Hooks E (2012) A green vehicle routing problem. *Transportation Research Part E Logistics & Transportation Review* 48:100–114

Felipe A, no MTO, Righini G, Tirado G (2014) A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation*

Table 5 Results by ALNS on 56 large scale instances

Instances	cost	m	f	rf	rm	time
C101_21	15911.04	12	1534.66	0.00	536.66	206.41
C102_21	14586.94	11	1426.724	0.00	489.11	219.72
C103_21	13818.97	11	1249.619	0.00	363.85	232.39
C104_21	11341.46	10	1107.084	110.26	51.46	227.76
C105_21	13829.97	11	1301.279	0.00	357.79	214.33
C106_21	14547.1	11	1461.816	0.00	475.53	215.20
C107_21	14400.56	12	1440.933	71.79	223.72	197.31
C108_21	13914.87	11	1315.205	0.00	372.54	225.88
C109_21	13960.37	11	1273.44	0.00	388.32	215.13
C201_21	7892.199	5	905.8217	0.00	433.51	249.93
C202_21	7883.339	6	1151.187	388.83	52.50	590.14
C203_21	8398.342	4	1100.417	0.00	703.60	265.97
C204_21	7643.746	4	952.9521	0.00	576.28	291.67
C205_21	7337.032	4	923.6695	0.00	519.62	270.71
C206_21	6990.417	5	1172.968	584.47	0.00	357.09
C207_21	7619.479	4	969.9159	0.00	568.71	276.31
C208_21	7133.712	4	880.4113	0.00	485.88	443.08
R101_21	20695.86	17	1794.048	0.00	452.12	101.74
R102_21	20626.17	17	1692.794	0.00	454.39	103.42
R103_21	18359.8	16	1711.043	174.03	135.54	120.05
R104_21	14043.41	11	1238.724	0.00	410.49	146.04
R105_21	17455.39	14	1474.197	0.00	455.21	140.46
R106_21	16295.01	13	1397.806	0.00	435.35	152.84
R107_21	14891.26	12	1228.76	0.00	381.65	171.51
R108_21	13932.11	11	1265.694	0.00	383.91	175.25
R109_21	15384.94	12	1390.005	0.00	454.59	169.32
R110_21	15203.06	12	1351.68	0.00	424.34	146.60
R111_21	13944.39	11	1269.53	0.00	385.75	155.56
R112_21	13695.9	11	1217.447	0.00	344.39	142.88
R201_21	7164.88	4	1737.489	986.05	0.00	405.38
R202_21	5988.215	4	1412.137	463.44	4.86	862.57
R203_21	5900.115	4	1251.152	499.55	0.00	903.81
R204_21	4507.271	3	1050.62	272.32	35.32	893.54
R205_21	5558.211	3	1396.986	800.35	0.00	890.88
R206_21	5466.247	3	1276.002	692.54	39.78	550.32
R207_21	4642.09	3	1186.485	384.95	0.00	341.90
R208_21	4744.102	3	1123.64	469.55	0.00	461.47
R209_21	5582.292	3	1309.109	713.47	50.15	597.51
R210_21	6425.375	4	1453.092	701.61	0.00	395.74
R211_21	4200.502	3	996.8906	223.88	0.00	1843.18
RC101_21	19945.56	16	1922.142	0.00	481.57	102.53
RC102_21	17351.2	14	1652.648	0.00	405.82	115.83
RC103_21	16622.53	13	1501.984	0.00	484.19	115.66
RC104_21	13631.33	11	1281.559	0.00	321.22	127.87
RC105_21	17330.12	14	1643.391	0.00	403.08	103.09
RC106_21	17113.44	13	1666.682	0.00	556.02	131.12
RC107_21	15631.16	12	1490.936	0.00	487.68	130.69
RC108_21	13844.69	11	1343.72	0.00	353.94	123.59
RC201_21	8485.253	5	2063.543	1002.28	6.06	468.08
RC202_21	7701.44	4	2048.307	848.79	106.99	237.30
RC203_21	7001.066	4	1735.573	895.89	0.00	394.45
RC204_21	6505.448	4	1363.94	785.72	0.00	852.59
RC205_21	7910.893	5	1741.124	741.84	36.54	451.80
RC206_21	7066.603	4	1786.184	846.27	22.87	820.13
RC207_21	6824.164	4	1627.745	741.30	37.53	544.36
RC208_21	6712.686	4	1334.503	535.89	136.10	480.41

Research Part E: Logistics and Transportation Review 71:111–128

Ghilas V, Demir E, Woensel TV (2016) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research* 66:170–180

Goeke D, Schneider M (2015) Routing a mixed fleet of electric and conventional vehicles. *European Journal of Operational Research* 245(1):81–99

Gonçalves F, Cardoso S, Relvas S, Barbosa-Póvoa A (2011) Optimization of a distribution network using electric vehicles: A vrp problem. In: 15th Congresso nacional da Associação Portuguesa de Investigação Operational, pp 18–21

Hiermann G, Puchinger J, Ropke S, Hartl RF (2016) The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research* 252:995–1018

Keskin M, Çatay B (2016) Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies* 65:111–127

Li B, Krushinsky D, Woensel TV, AReijers H (2016) An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research* 72:12–30

Maximilian S, Grit W (2018) An adaptive large neighborhood search for the location-routing problem with intra-route facilities. *Transportation Science* 52(2):331–352

Montoya A, Guéret C, EMendoza J, GVillegas J (2017) The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological* 103:87–110

Paz JC, Granada-Echeverri M, Escobar JW (2018) The multi-depot electric vehicle location routing problem with time windows. *International Journal of Industrial Engineering Computations* 9:123–136

Pelletier S, Jabali O, Laporte G (2016) Goods distribution with electric vehicles: Review and research perspectives. *Transportation Science* 50:3–22

Penna PHV, Afsar H, Prins C, Prodhon C (2016) A hybrid iterative local search algorithm for the electric fleet size and mix vehicle routing problem with time windows and recharging stations. *IFAC-PapersOnLine* 49(12):955–960

Renaud M, Fabien L, Olivier P (2013) An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science* 47(3):344–355

Sassi O, Cherif WR, Oulamara A (2015) Vehicle routing problem with mixed fleet of conventional and heterogenous electric vehicles and time dependent charging costs. In: *EvoCOP 2015: Evolutionary Computation in Combinatorial Optimization*, pp 138–149

Schiffer M, Walther G (2017) The electric location routing problem with time windows and partial recharging. *European Journal of Operational Research* 260:995–1013

Schneider M, Stenger A, Goeke D (2014) The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science* 48:500–520

Shaw P (1997) A new local search algorithm providing high quality solutions to vehicle routing problems. Tech. rep., Department of Computer Science, University of Strathclyde, Scotland

Solomon M (1987) Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* 35(2):254–265

Stefan Ropke DP (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40(4):455–472

Yang J, Sun H (2015) Battery swap station location-routing problem with capacitated electric vehicles. *Computers and Operations Research* 55:217–232

Yossiri A, Jean-François C, Raf J (2014) Optimization-based adaptive large neighborhood search for the production routing problem. *Transportation Science* 48(1):20–45

MISTA 2019

The video sequencing problem

Hatice Çalık* · **Aäron Hallaert** · **Farzaneh Karami** · **Greet Vanden Berghe**

Abstract We introduce a video sequencing problem that requires selecting video clips and assigning them to the ordered segments of a final video to be formed. Each segment has a set of criteria, which must be satisfied by the assigned clips and each clip-segment assignment has a nonnegative score. The objective is to maximize the total score of the ordered selection of clips. The motivation behind studying this problem is to support the automatization of the video editing process, which might be extremely time consuming otherwise. For example, a news broadcast requires cutting and combining a huge number of video clips in a very short amount of time, which is difficult to achieve via a manual process. We provide an integer programming formulation for this problem and discuss the similarities and differences of the proposed video sequencing problem with those found in the literature. We also develop a local search-based heuristic and conduct a computational study on the two methods using data generated by a recognition software package.

1 Introduction

Recent advances in technology have made it very easy to collect and store a tremendous number of video clips. Managing this large quantity of raw footage and converting it into finished videos is a cumbersome challenge because selecting and editing video material is very time-consuming. Some of these clips are used to compose larger videos and such a process requires a lot of work, which includes viewing and annotating clips

Hatice Çalık
*Corresponding author.
KU Leuven, Department of Computer Science, CODeS, Belgium
E-mail: hatice.calik@kuleuven.be

Aäron Hallaert
KU Leuven, Belgium

Farzaneh Karami
KU Leuven, Department of Computer Science, CODeS, Belgium

Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS, Belgium
E-mail: greet.vandenberghe@cs.kuleuven.be

manually as well as cutting and fitting them into the final video. Such a task can make good use of automatization.

This study is conducted in cooperation with a company which offers support in managing video data through an online platform. This platform enables users to manually add *metadata* to videos. 'Metadata' is the collective term for certain annotations (labels) which can be related to video content (for example an interview), faces, objects, speech transcriptions or audio. It registers and quantifies the human perception of a video and makes it quicker and easier to search a video collection.

In the traditional video editing sector, the video archive is managed manually by appointing a person to watch every video and add metadata to the file. Manually completing this task is usually inefficient and often leads to inaccuracies. Consider the process of adding a face annotation to the metadata of a video. Information such as when this recognition occurs and how long it is visible is usually not specified since adding these timestamps would delay the entire process. The face-time percentage of an annotated person in a long video may be very low. To actually retrieve the fragment where the desired person is in frame, a video editor still needs to watch the whole video, searching for this specific person. This is a very time consuming process.

Artificial intelligence can help with automating this annotation task. Several globally-known software companies already have software packages to recognize objects and faces. These software packages can complete the annotation task much more precisely and faster than a human. Such software is essential to automatically gather metadata of individual video clips.

The focus of this paper is to introduce the video sequencing problem (VSP). The VSP consists of selecting and ordering a subset of given video clips with metadata by satisfying user-specified requirements related to the content of the resulting video. User specifications, which we also refer to as the criteria, may be face or speech related. Users can specify when they want a certain face to occur in the resulting video, if there have to be other faces in the same frame or if someone has to talk over a certain part of the video. Therefore, we consider clips with metadata containing *face and speech recognitions*. Every *recognition* has a time frame in which it occurs, a type and a description (for example, the name of the person related to this recognition). Recognitions are used to search video fragments for the criteria specified by the user. For each criterion, all the clips that meet the requirements are collected as potential clips for relevant segments via a preprocessing procedure. The objective is to maximize the total score of the selected clips, which is considered as the duration of relevant recognitions based on the requirements of the user. The output is a sequence of selected video clips where each clip belongs to at most one segment, all criteria are met for each segment and the total duration is within predefined boundaries. We provide an integer programming formulation as well as a heuristic approach for the VSP.

The remainder of this paper is organized as follows: Section 2 summarizes related work in the metadata collection and video editing literature. Section 3 formally describes the VSP with an integer programming (IP) formulation and discusses its similarities and differences with several relevant problems from the operations research literature. Section 4 initially details the preprocessing procedure to collect potential clips to be used by the IP model as well as by the local search-based approach, which is also detailed in this section. Section 5 presents the results of the computational study comparing the IP formulation and the heuristic method for several problem instances. Finally, Section 6 concludes with some suggestions concerning future research directions.

2 Related work on metadata collection and video editing

Video editing requires several inputs, including the original video files and metadata, which play a crucial role in the automatization of the process. Among the methods available for metadata collection, Ahanger and Little (1998) add metadata to a video by manually placing every input clip in the right segment. For example, video clips where two people greet each other can be a potential clip for the start segment, conversations for the middle segment and farewells for the end. This approach still requires a person to watch all the input clips and manually group them for the correct segment.

Metadata can also be collected based on technical information instead of content. This technical information may consist of timestamps indicating when the video was filmed. These timestamps can be used to specify the order in which the input clips will occur in the resulting video (Ahanger and Little, 1998). Where a shot change is detected or where certain audio clips start and stop can also be useful when editing videos (Casares et al., 2002).

A more advanced way to gather metadata is introduced by Arev et al. (2014). Synchronized video clips of the same event are used to make a 3D-reconstruction which is useful to find the event's point-of-interest. In this manner it is possible to rate each input clip based on the camera direction.

In the montage phase, the structure of a desired video is often defined by certain limitations and assumptions. One key restricting component is the video context, examples of which include: a news broadcast, a reality-show, a documentary or a thrilling action scene. While an action requires a large number of fast consecutive clips, a nature documentary needs long and slow fragments. Some video editing algorithms focus on a script (Ahanger and Little, 1998), others do not and therefore it might be necessary to make assumptions on the structure of the resulting video or to let the algorithm decide (Arev et al., 2014). Chen et al. (2012) select and montage video material based on visual similarity. They try to make a smooth visual summary, while Ahanger and Little (1998) focus on providing a smooth transition related to the content.

In addition to content-based video editing, technical video editing can also be automated. Casares et al. (2002) discuss a technical montage in which video clips are cut on the basis of a localization of audio clips and shot change detections.

A crucial part of the video editing process is collecting potential clips. One should decide which clips are too long, which are better than the others, and whether or not they satisfy user specifications. Ahanger and Little (1998) suggest two ways of selecting the potential clips: 'boolean metrics' and 'ranking metrics'. Boolean metrics select all the clips that meet the demands of the user. Ranking metrics rank all the input clips based on how well they meet the requirements.

Once the suitable clips have been collected, the final video can be obtained by filling the available time budget. This procedure is usually not completely random and takes the user-specified constraints, assumptions, context or content similarity into account. In certain cases, cutting the input clips may increase the possibility of obtaining higher quality outputs in this assembly process. Casares et al. (2002) discuss different techniques to cut clips. Audio-based cuts make sure the whole required audio fragment is present with cuts taking place at shot change. Video-based cuts ensure the required video fragment is in the cut and the audio is not abruptly interrupted.

All aforementioned papers provide interesting technology to support video editing. We will focus on a process step which has not been automatized, yet offers a huge potential for efficiency improvement.

3 Mathematical formulation

Given a set of video clips $V = \{1, \ldots, k\}$ with t_v being the length of clip $v \in V$, the VSP requires selecting a subset $\bar{V}$ of V to obtain a video whose length $T = \sum_{v \in \bar{V}} t_v$ is between L and U time units ($L < U$, $T \in [L, U]$). This final video should consist of a set of segments $S = \{1, \ldots, n\}$ where each segment $s \in S$ has a set of criteria C_s which must be satisfied by the selected clips in $\bar{V}$. Let $V_c \subset V$ be the set of clips satisfying criterion $c \in C_s$ and $p_{vc} \geq 0$ represent the satisfaction value (score) of clip $v \in V$ for criterion c, the objective is then to maximize the total score of the clips in $\bar{V}$.

Define x_{vs} as a binary decision variable that takes value 1 if clip $v \in V$ is selected for segment $s \in S$, and 0 otherwise. The following is an IP model for the VSP.

$$\max \quad \sum_{s \in S} \sum_{c \in C_s} \sum_{v \in V_c} p_{vc} x_{vs} \tag{1}$$

$$\text{s.t.} \quad \sum_{v \in V_c} x_{vs} \geq 1 \qquad \forall s \in S, c \in C_s \tag{2}$$

$$\sum_{v \in V} \sum_{s \in S} t_v x_{vs} \geq L \tag{3}$$

$$\sum_{v \in V} \sum_{s \in S} t_v x_{vs} \leq U \tag{4}$$

$$\sum_{s \in S} x_{vs} \leq 1 \qquad \forall v \in V \tag{5}$$

$$x_{vs} \in \{0, 1\} \qquad \forall v \in V, s \in S \tag{6}$$

The objective function (1) maximizes the total score of the selected clips. Constraints (2) select at least one clip for each criterion of each segment. Constraints (3) and (4) ensure that the total duration of the final video is within the predefined boundaries. Constraints (5) make sure that the same video clip does not appear in multiple segments. Finally, Constraints (6) are binary restrictions.

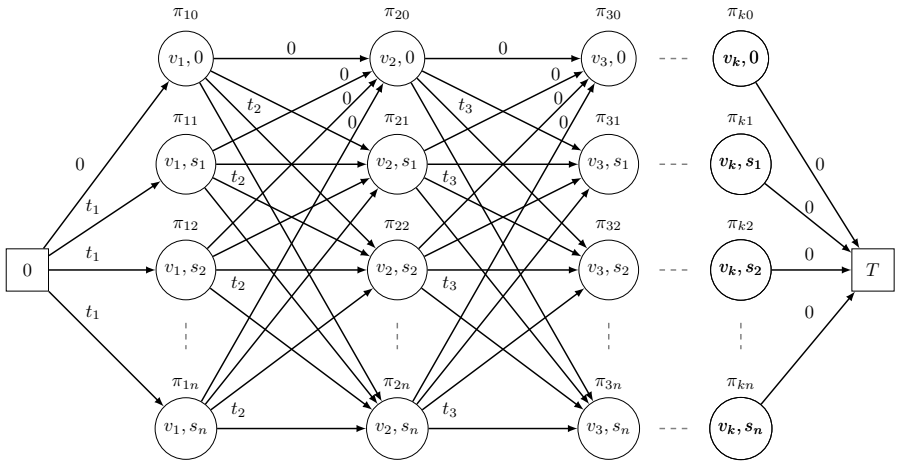


Fig. 1: A network representation with clip-segment assignments.

The VSP can be represented on a directed network such as that depicted in Figure 1. In this network, node (v_i, s_j) denotes the assignment of clip v_i to segment s_j and $(v_i, 0)$ denotes the case where clip v_i is not assigned to any segment and thus not used at all. The length of an arc represents the amount of time used. The length of arcs entering node $(v_i, 0)$ is equal to zero whereas it is equal to t_{v_i} for arcs entering node (v_i, s_j), where $v_i \in V$, $s_j \in S$. The score π_{ij} of visiting node (v_i, s_j) is equal to $\sum_{c \in C_{s_j} : v_i \in V_c} p_{vc}$ for $s_j \in S$ and $\pi_{i0} = 0$ for $(v_i, 0)$, $v_i \in V$.

Figure 2 provides a more informative network that also indicates criteria satisfaction. In this figure, we consider a sample problem instance with four video clips $\{v_1, \ldots, v_4\}$ and three segments each of which has three criteria to be satisfied. More specifically, $C_1 = \{c_1, c_2, c_3\}$, $C_2 = \{c_4, c_5, c_6\}$ and $C_3 = \{c_7, c_8, c_9\}$. If clip v_i satisfies criterion c_j, we add a node (v_i, c_j) with score p_{ij} to the network. The VSP then requires to find a maximum score path from node 0 to node T that visits at least one node for each criterion and respects the total duration limits (3) and (4).

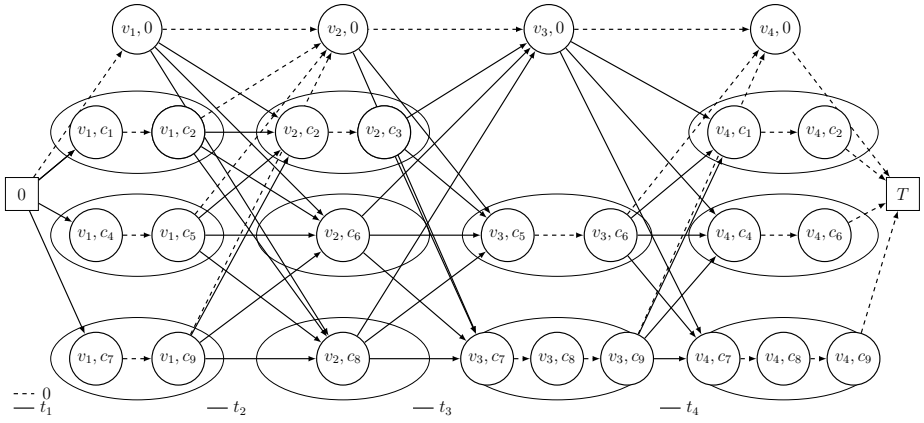


Fig. 2: A network representation with clip-criterion assignments for an example of four video clips and three segments, each with three criteria. The score of node (v_i, c_j) is equal to p_{ij}. The length is equal to zero for the dashed arcs and equal to the duration of the corresponding video clip for entering solid arcs.

The VSP contains many components similar to those present in several well-studied problems from the literature. When omitting Constraints (2) and (3), one can represent this relaxed problem as an *orienteering problem* (OP) from node 0 to node T on either of the (acyclic) networks visualized in Figures 1 or 2. The OP aims to find a path on a given network from a source node to a destination node such that the total length of the path is less than a certain threshold and the total score collected from visited nodes is maximized. For a formal description of the OP, its complexity and reviews of its variants, we refer to Vansteenwegen et al. (2011). An extended survey containing relatively recent OP variants, solution methods and applications is provided by Gunawan et al. (2016).

The most relevant OP variant is probably the split delivery capacitated team orienteering problem with minimum delivery amounts (SDCTOP-MDA) (Vansteenwegen and Gunawan, 2019). In the context of tourism applications where the OP aims to

provide a visit plan, there may be lower and upper limits on the number of certain types of attractions to visit, for example at least one church and at most two museums. The VSP can be considered as a variant of the capacitated OP with minimum delivery amounts (COP-MDA) considering the criteria satisfaction constraints (2) and no-repetition constraints (5), which have a slightly different structure than the capacity constraints in the SDCTOP-MDA.

Video selection in the VSP is also similar to selection in the well-known knapsack problems. Note that the relation between the node selection in the OP and knapsack problems is already noted by Vansteenwegen and Gunawan (2019). If one considers a knapsack with multiple segments of flexible size and items yielding different values in different segments, then finding a maximum value assignment of items to segments becomes a relaxation of the VSP without Constraints (2). To the best of our knowledge, no such knapsack variant has been studied in the literature.

Although there is an upper limit on the total length of the final video in the VSP, there is no upper limit on the length of the individual segments. This property resembles the flexible compartment variant of the multi-compartment vehicle routing problem introduced by Henke et al. (2015). Even though the two problems share some properties, they differ in many aspects. The most fundamental difference is that the VSP has more of a maximal covering structure which selects the most profitable subset of items fitting in the time budget while the problem by Henke et al. (2015) aims to meet all the demand at minimum cost.

The criteria satisfaction constraints (2) of the VSP can also be associated with the well-known set covering problem. Considering this covering aspect and a routing structure similar to the OP, the VSP also shares certain features with the covering tour problem by Gendreau et al. (1997). Recalling Figure 1, the covering tour problem requires visiting a subset of nodes such that all the criteria are covered (i.e. satisfied by at least one clip) and the total length of the tour is minimized. Unlike the covering tour problem, the VSP aims to maximize the total score of the selected nodes while ensuring the total length of the tour (final video) is within certain limits.

The following section describes a heuristic which is combined with the preprocessing procedure that enables searching for video clips satisfying multiple user criteria such as face-face, face-speech, speech-speech recognitions as well as cutting clips into smaller pieces. Note that this preprocessing procedure is utilized also for obtaining the inputs of the IP model.

4 Heuristic approach

4.1 Preprocessing for multiple criteria search on clips

When searching for clips which meet certain criteria where simultaneous face or speech segments are needed, the set of recognitions belonging to a clip must be iterated too many times. Consider the example where person A needs to be in the frame while at the same time person B is speaking. If the recognition of Face A in a clip is found, it has to be iterated over all other recognitions, searching for person B who is speaking at the same time as the recognition of person A. This process can be accelerated using a binary interval tree of recognitions in a clip. If person A is recognized in the clip, all overlapping recognitions are searched in logarithmic time in relation to the number of recognitions in the tree.

Figures 3 and 4 illustrate this tree search and how the nodes of the tree are pruned in logarithmic time. In this example, we have a clip of Face A whose time interval is $[25.0, 36.0]$. The seven nodes of the tree in Figure 4 represent the time frames of the available recognitions. At the first iteration of the logarithmic search, a branch of the tree is cut since the maximum end value of its children is greater than the beginning time of Face A clip. Therefore, only four nodes are explored and three of them (in bold) represent the overlapping clips.

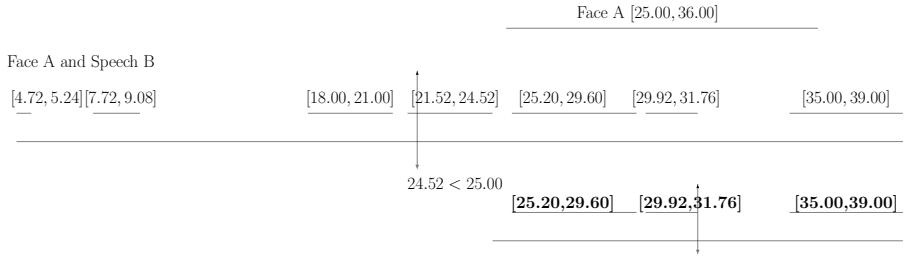


Fig. 3: Logarithmic pruning of time intervals.

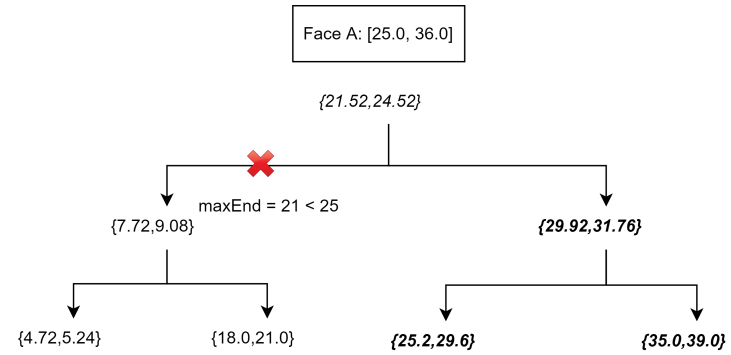


Fig. 4: Using binary interval trees for searching overlapping recognitions with respect to the recognition of Face A.

After collecting all overlapping clips, we search in this set for a recognition where person B is speaking. If this recognition is in the set of overlapping clips, the requirement is met and the clip is selected as a potential clip for the segment of this criterion.

When the input clips can be cut, all the recognitions are searched in a similar manner and cuts are made based on these recognitions. The first recognition in time that has a role in fulfilling the requirement determines the start of the cut and the last recognition in time determines its end. In doing so, the irrelevant frames can be avoided in the final video. Once a cut is made, it is not possible to cut this sub-clip into smaller pieces to satisfy another requirement. In a similar fashion these sub-clips are chosen as potential clips for the corresponding segment.

4.2 The local search algorithm

The initial solution for the local search is constructed by selecting a random candidate clip from V_c for every criterion $c \in C$. The initial solution $\bar{V}$ does not have to be feasible.

The local search strategy explores the solution space to improve this initial solution by adding, deleting or both adding and deleting the potential video clips. The local search method contains a very simple step-by-step procedure consisting of two parts that realize these modifications. At each iteration, a random video clip v from the candidate videos is added to $\bar{V}$. If the resulting video is longer than the duration limit U after adding v, a number of random clips are deleted from $\bar{V}$. The number of clips to be removed is a parameter which can be specified by the user.

The acceptance criteria are described in Algorithm 1. The current solution is accepted when it is better than the best solution, but it can also be accepted if both the best and current solution are infeasible. If the best solution is feasible and the current is not better than the best, there is still a chance that the current solution will be accepted for the next iteration. Although the thresholds of this chance can be specified a priori, preliminary experiments showed that the thresholds employed in Algorithm 1 performed the best.

```
Data: Current solution and next, best and initial solution
if current.cost > best.cost & current.isFeasible then
    next= current;
    best= next;
else if !best.isFeasible then
    next=current;
else
    generate r1 = rand(0, 1);
    if r1 > 1/9 then
        next= initialSolution;
    else
        generate r2 = rand(0, 1);
        if r2 > 1/3 then
            next = current;
    end
end
```

Algorithm 1: Acceptance function.

5 Computational study

The IP model and the local search algorithm were implemented in Java and the experiments were run on an Intel® Core™ i7-4720HQ CPU, 2.60 GHz computer with 12 GB of RAM. In order to solve the IP model, ILOG IBM CPLEX 12.9 was used. The methods were applied to video material from 'De Wereld Rond met 80-jarigen' (a production of VTM) and 'WTFock' (a production of Sputnik Media). The raw video material was annotated using a recognition software provided by Valossa. The recognition specifications of these VSP instances are denoted by the following notation:

– F: the end result must contain a fragment where a single face must be present.

– S: the end result must contain a fragment where a single speaker must be present.
– FF: the end result must contain a fragment where multiple faces must be simultaneously present.
– FS: the end result must contain a fragment where a face and speaker must be simultaneously present.
– C: the input clips can be cut to be part of the resulting video.

Further specifications of the instances will be shared during the conference presentation. These instances are available upon request.

Table 1 details the results obtained when applying the IP model and the heuristic to problem instances which only consider single face recognitions. For each of these instances, the heuristic was only run once. Although the heuristic can find optimal solutions for only two of the five instances, its solutions for the others are not considerably inferior compared to the optimal solutions. The largest gap between the optimal and the heuristic solution value is 5.5%.

Table 1: IP and heuristic results for the problems with only 'F' criterion.

	IP Model		**Heuristic**	
Specifications	**Optimal Score**	**Duration (T)**	**Best Score**	**Duration (T)**
L=8,U=12 seconds	11.44	11.92	11.44	11.92
L=24,U=36 seconds	33.32	35.84	31.68	36
L=20,U=30 seconds	21.24	23.96	21.24	23.96
L=24,U=36 seconds	35.84	35.76	33.96	34.84
L=48,U=72 seconds	96.52	71.96	94.72	71.8

Table 2 provides the results for the problem instances with multiple criteria as well as the possibility of clip cuts. The heuristic is again able to find optimal solutions for two of these instances. Among them, for the instance at the bottom row of the table, the IP solution has a longer duration than the solution of the heuristic. This is because the heuristic uses dummy clips (the clips with $p_{vc} = 0$ for all criteria) only when they are needed, that is, for satisfying the minimum length requirement. On the other hand, the IP model might provide alternative solutions including these dummy clips even when they are not needed. If dummy clips are undesirable for the user, one can easily remove these clips from the set of potential clips during the preprocessing procedure.

Both methods' computational times remain under 20 seconds when solving any instance in these tables. We observe that the size of these instances are too small to challenge the mathematical model and more extensive experiments on large scale instances need to be conducted to see the limits of the IP model.

6 Conclusion and future work

This paper introduced a video sequencing problem (VSP) and provided optimization-based methods to support the video editing process. The aim is not to take over the complete task but to support its most important parts, such as the selection and placement of video clips. Recognitions, obtained by external software, make it possible

Table 2: IP and heuristic results for the problems with multiple criteria and cuts

	IP Model		Heuristic	
Specifications	**Optimal Score**	**Duration**	**Best Score**	**Duration**
L=48, U=72 seconds FF (Vidal - Rene), FS (F: Aureel - S: Sieg), F (Amanda), C	56.64	71.92	40.12	49.6
L=72, U=108 seconds FF (Sieg -Olga), F (Amanda), F (Willy), C	34.64	94.44	34.16	82.04
L=32, U=48 seconds FF (Vital - Rene), FF (Sieg - Olga), F (Mariette), F (Aureel)	44.32	47.76	44.32	47.76
L=64, U=96 seconds FS (S: Sieg - F: Aureel), FF (Olga - Sieg), F (Amanda), S (Willy), C	66.28	88.36	56.48	66.96
L=24, U=36 seconds F (Amanda), F (Olga), F (Willy), C	32	35.32	32	32

to search for and identify the video material that satisfies desired specifications aided by the binary interval tree. When possible they are cut out of the input clips in order to meet the user's requirements.

Although the main focus of this paper is to introduce a new sequencing problem with an integer programming formulation, we also include a local search-based heuristic which is rather easy to combine with the preprocessing of input materials as it does not require external mathematical solvers.

An obvious extension is to add other recognitions such as object and audio recognitions. These can be implemented very easily since they will behave analogous to the current face and speech recognitions.

Introducing greater flexibility for choosing requirements can be realized by specifying the duration of individual segments or letting the user choose between new features. The more possibilities of specifying the end result, the more scenarios will benefit from the presented approach to automate the video editing process.

It would also be interesting to consider smooth transition based on content to make a more pleasing video. In order to do this, one could cluster clips based on their visual similarity and/or audio similarity, that is, those clips in which the same keywords are spoken. This would, however, require more detailed annotations than we employed for the VSP.

Acknowledgements The video sequencing problem was introduced to KU Leuven by Limecraft. Research partially supported by Data-driven logistics (FWO-S007318N). Editorial consultation provided by Luke Connolly (KU Leuven).

References

Ahanger, G. and Little, T. D. (1998). Automatic composition techniques for video production. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):967–987.

Arev, I., Park, H. S., Sheikh, Y., Hodgins, J., and Shamir, A. (2014). Automatic editing of footage from multiple social cameras. *ACM Transactions on Graphics (TOG)*, 33(4):81.

Casares, J., Long, A. C., Myers, B. A., Bhatnagar, R., Stevens, S. M., Dabbish, L., Yocum, D., and Corbett, A. (2002). Simplifying video editing using metadata. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 157–166. ACM.

Chen, Q., Wang, M., Huang, Z., Hua, Y., Song, Z., and Yan, S. (2012). Videopuzzle: Descriptive one-shot video composition. *IEEE Transactions on Multimedia*, 15(3):521–534.

Gendreau, M., Laporte, G., and Semet, F. (1997). The covering tour problem. *Operations Research*, 45(4):568–576.

Gunawan, A., Lau, H. C., and Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332.

Henke, T., Speranza, M. G., and Wäscher, G. (2015). The multi-compartment vehicle routing problem with flexible compartment sizes. *European Journal of Operational Research*, 246(3):730–743.

Vansteenwegen, P. and Gunawan, A. (2019). *Orienteering Problems Models and Algorithms for Vehicle Routing Problems with Profits*. Springer, Switzerland.

Vansteenwegen, P., Souffriau, W., and Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10.

MISTA 2019

Facilitating same-day delivery through conveyor operations modeling and optimization in distribution centers

Farzaneh Karami · Wim Vancroonenburg · Greet Vanden Berghe

the date of receipt and acceptance should be inserted later

Abstract Optimizing conveyor operations concerns the problem of transferring a given number of items among multiple loading and unloading locations within a minimum makespan. Achieving this requires solving a multi-commodity quickest flow problem which is adapted to accommodate real-world conveyor operation constraints. Something which is currently missing from the literature. To address this gap, this study compares two path-based formulations for routing and scheduling items on the conveyor network. The first concerns the transshipment of items on a single path during a given time horizon with the objective being to minimize the makespan. The second formulation targets the same objective through utilizing multiple paths. A heuristic algorithm is proposed for solving the multi-path-based formulation given the need for high quality solutions within reasonable computational times. Experiments are conducted on a set of benchmark instances and the results provide evidence concerning how controlling the number of used paths enables handling real-world levels of complexity.

1 Introduction

Offering same-day delivery (SDD) concept is a recent trend in the e-commerce industry, and one enabled by some online logistic service providers. SDD is becoming very common and the service has led to an ever-increasing freight volume, all of which requires sorting, packing and transportation under tight deliver schedules (Wahba, 2015). The time horizon available to service all SDD requests is primarily devoted to the transportation, with only a very narrow time window left for sorting and packing. Therefore, a high capacity and efficient distribution center (DC) is a prerequisite for truly realizing SDD. DCs serve as intermediary storage centers for goods in-between two stages of a supply chain. Automation is a solution for

F. Karami[1] (corresponding author), W. Vancroonenburg[12], G. Vanden Berghe[1]
[1] KU Leuven, Department of Computer Science, CODeS & imec, Gebroeders De Smetstraat 1, 9000 Gent, Belgium
Tel.: +32 9 265 87 04
E-mail: {farzaneh.karami, wim.vancroonenburg, greet.vandenberghe}@cs.kuleuven.be
[2] Postdoctoral research fellow at Flanders Research Foundation - FWO

meeting SDD needs and increasingly more DCs are being automated to increase throughput. These DCs mainly consist of storage areas, each of which is equipped with a robot which automatically collects and loads items onto the conveyor. DCs typically employ conveyor networks to transport items from the storage areas to the packing areas. Each conveyor consists of a set of linked segments arranged in a closed-loop belt, along which there are multiple loading and unloading locations. Each segment has a limited capacity and takes a certain period of time to transport items. To optimize conveyor operations, the problem of transferring a given number of items from the loading to the unloading locations within a minimum makespan must be solved. Makespan denotes the time between the start of the planning horizon and the last items' arrival time at its unloading location.

Boysen et al. (2018a) conducted a comprehensive survey of the scientific literature concerning all kinds of fully-automated conveyor-based sorting systems from an operational research perspective. All of the proposed methods assume one of two major approaches: simulation-based and mathematical formulation-based. For example, Russell and Meller (2003) proposed a descriptive model for conveyor operations and conducted a simulation study which included stochastic aspects to verify their results. Jarrah et al. (2014) considered the distribution center of a postal service provider as a hierarchical system. By doing so, they decompose conveyor operations into smaller subsystems to handle. Boysen et al. (2018b) introduced a mathematical model for conveyor operations and proposed and validated priority rule- and dynamic programming-based methods. They investigated the relationship between sequencing and sorting operations and suggested that sequence optimization would help prevent congestion at loading locations. Banerjee et al. (1999) considered conveyor operations as a transshipment problem where the goal is to minimize the total travel time.

Quickest commodity flow-over-time approaches are among the most suited to provide a modelling representation of conveyor operations. Each commodity is defined as a triple of its loading location, unloading location and amount of flow demanded. For example, the quickest path problem (QPP) requires loading-unloading flow to be routed on a given network while taking into account the capacity of arcs, while minimizing the makespan. Since a conveyor has multiple loading and unloading locations, its operations are closely related to the multi-commodity QPP and solving it is equivalent to solving the maximum flow-over-time problem. While there is substantial literature on the static multi-commodity QPP, hardly any results on multi-commodity QPP with flow-over-time are available. Melchiori and Sgalambro (2018) have illustrated that the multi-commodity QPP with flow-over-time is NP-hard. Therefore, equipping a conveyor with multiple loading and unloading locations necessarily introduces an additional layer of complexity to the routing and scheduling of its flows.

The concept of flow-over-time was originally introduced by Ford and Fulkerson (1958). They extended the static flows problem to a flow-over-time context by demanding the completion of an optimal routing within a given fixed time horizon. They introduced a general procedure to translate a flow-over-time problem to a static one by means of a time-expanded graph. This enabled taking advantage of a polynomial algorithm developed for the static problem. Ford and Fulkerson

showed that an optimal solution for the maximum flow-over-time problem was easily attained by decomposing it into flow on paths. While this seems practical in the case of the single commodity flow problem, it demands some adjustment for the multi-commodity flow problem.

The QPP forces flow to be routed on a single path (Pascoal et al., 2006). Since this constraint will impair solution quality in many real-world applications, the literature has suggested increasing the number of paths to a reasonable fixed number. The problem of ranking k-quickest paths has been addressed by Pascoal et al. (2007) and applied to the routing of data packets across the internet by Clímaco et al. (2007). The literature on multiple paths has demonstrated that adjusting the fixed number of paths helps in many real-world applications. For example, the emergency transportation problem has been solved by Melchiori and Sgalambro (2018) by minimizing the makespan of transshipment operations while imposing restrictions on the number of paths for each commodity. Such a modeling approach is new for conveyor operations. Furthermore, assuming a fixed number of paths for each commodity without considering the flow demands of each is not realistic in a conveyor operation context.

A few of the unaddressed question have motivated the present paper. Is it better to employ a single path for each commodity or split each commodity's flow across several paths? Does controlling the number of paths used per commodity help? How does a realistic restriction on the number of paths utilized affect solution quality? And, finally, is imposing a realistic number of paths demands an essential modeling requirement which is not captured by the multi-commodity QPP? To address these, two path-based formulations for the routing and scheduling of commodities' flow are proposed and compared: the single-path-based multi-commodity QPP and the multi-path-based multi-commodity QPP. Finally, a local search-based matheuristic is proposed for solving the multi-path-based multi-commodity QPP.

2 Problem description

Figure 1 depicts a simple example of a DC's conveyor belt and provides an illustrative overview of items' flow. This conveyor has two loops and consists of two loading locations, L_1 and L_2, and two unloading locations, U_1 and U_2.

The conveyor network can be represented as a directed graph $G = (V, A)$ where V is the set of nodes and A is the set of arcs. G can be constructed by considering each junction as a node and the conveyor segment that links any two connected nodes as an arc. These arcs have limited capacities and take some time to transit on each of them. For each arc $(i, j) \in A$, parameters c_{ij} and λ_{ij} denote its capacity and travel time, respectively. Loading and unloading locations of a conveyor correspond to sources and sinks in this corresponding graph. It is assumed that a set of commodities $\mathbb{C}$ is given, where a commodity $c \in \mathbb{C}$ is defined by a triple (L_c, U_c, σ_c) which corresponds to its loading location, unloading location and flow demands. A path p from $i \in V$ to $j \in V$ in G is defined as a sequence of the form $p = \langle i = v_0, v_2, ..., v_{l_p} = j \rangle$ and $(v_k, v_{k+1}) \in A$ for any $k \in \{0, ..., l_p\}$. The set of

paths for commodity c will be denoted by $\mathcal{P}_c$. Paths do always begin at a loading location and end at an unloading location.

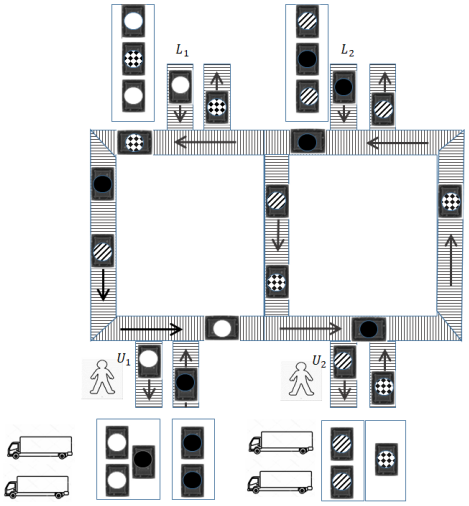


Figure 1: Schematic layout of a conveyor belt as a hatched area

A commodity flow's journey along a conveyor can be summarized as follows: First, they are brought from storage to their respective loading locations based on their predefined sequence. Second, a switch system successively loads them at the loading location onto the conveyor. Commodity flow then travel to predefined unloading locations. Finally, the flow which has left the conveyor is collected and packed by human workers at unloading locations.

As introduced by Ford and Fulkerson (1958), a conveyor commodity flow-over-time problem can be translated into a static flow problem on a time-expanded graph. To generate the time-expanded graph, first the entire scheduling horizon is divided into a finite number of intervals. The interval is calculated based on the shortest conveyor arc, l_{min}, and the conveyor velocity v ($\frac{m}{sec}$), as $t_{int} = \frac{l_{min}}{v}$. Capacity c_{ij} and transmission time λ_{ij} for each arc (i,j) with length l_{ij} can be calculated by: $c_{ij} = \lceil \frac{l_{ij}}{l_{min}} \rceil$ and $\lceil \frac{l_{ij}}{v} \rceil$, respectively. The time-expanded graph is a directed graph $G_{\hat{T}} = (V^{\hat{T}}, A^{\hat{T}})$ where $V^{\hat{T}}$ contains $\hat{T} = \lceil \frac{T}{t_{int}} \rceil$ replicas of the original graph's nodes, where $V_{\hat{T}} = \{(i,t) | i \in V,\ t \in \hat{T}\}$. This graph has two sets of arcs: $A_H = \{((i,t),(i,t+t_{int}))|i \in V, t = 0,...,T - t_{int}\}$, and $A_M = \{((i,t),\ (j,\hat{t}))|(i,j) \in A,\ \hat{t} = t + \lambda_{ij} \leq T\}$. A_H contains the holdover arcs which allow flow to wait at a single node. Meanwhile, A_M contains the set of arcs which connect separate nodes in the time-expanded graph, respecting the associated travel time. Note that $A_{\hat{T}} = A_M \cup A_H$. Since the only nodes with a capacity greater than zero are loading locations, the holdover arcs are only allowed at these locations. The time-expanded graph corresponding to the conveyor graph of Figure 2 which has seven time intervals (seven steps of expansion) is illustrated in Figure 3.

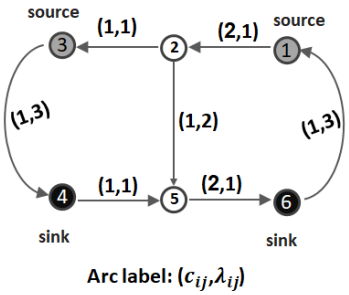


Figure 2: A conveyor network

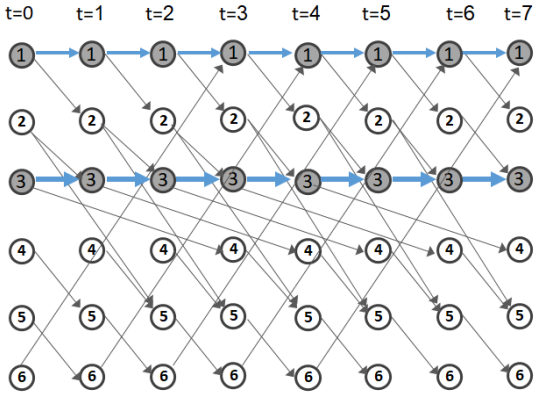


Figure 3: The conveyor graph expanded in time. The holdover arcs are highlighted in blue.

3 Solution approach

During the SDD horizon, customers' requests are continuously registered and initially listed based on their arrival time. In order to achieve optimal routing and scheduling and meet the SDD goal, two path-based formulations are introduced in this section. The aim of these formulations is to find routes whereby commodity flow items are shipped through either a single or multiple paths. Paths differ from each other with respect to at least one arc. In the multi-path-based formulation, paths are therefore not required to be completely different. The number of different paths used by each commodity c must not exceed a given parameter k_c^{max}. Both proposed formulations respect the conveyor's practical limitation: shared arc capacities must be respected and holdovers are only allowed at source nodes.

The single-path-based formulation In this formulation, the entire flow of each commodity must be transshipped through a single path. More specifically, it is assumed that a commodity's flow may wait at the loading location, but once the transshipment is started it is deterministically routed through the same unique path over consecutive time intervals. It is assumed that once a commodity's item is flowing on

the path, no other commodity's items can use this path. The path will be available only after the entire flow of the current commodity has been transported. Given these conditions, the commodities' flow at each loading location can be sequenced and split into subsets, each of which is scheduled at a distinct point in time. The first unit of flow leaving the loading location determines the path through which the entire commodity flow must be scheduled on consecutive uninterrupted time intervals. This prevents any possible interruptions and perturbations in the process while it is underway. Table 1 presents a list of constraints for the single-path-based formulation.

Table 1: The single-path-based formulation constraints

Constraint	Description
C1	Each commodity flow is not forced to start at time zero
C2	Each commodity flow transshipment is postponed by routing it through a holdover arc
C3	Each commodity flow will not split into subsets
C4	Flow arrivals at each junction must not take place simultaneously
C5	Flow conservation at all nodes
C6	Arc capacity constraints
C7	The flow demand of each commodity constrains total amount of items to be transshipped

When several flow items assigned to different paths share the same arc at the same time, this corresponds to a node in the time-expanded graph where flows coming from several arcs collide. This situation is a source of interruption in conveyor operations and one which violates an arc capacity. In this way, interruption of the conveyor operation is prevented. Figures 4 and 5 show the routing of commodity flow while considering these conveyor operational constraints. It is clearly seen that by considering that flow arrivals at each junction must not take place simultaneously the makespan will extended by two time intervals.

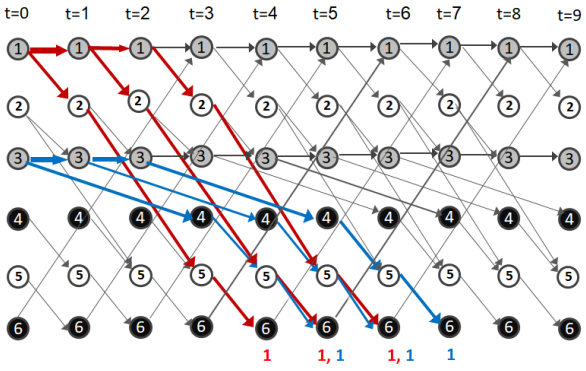


Figure 4: Routing c_1(1,6,3) on path 1-2-5-6 and c_2(3,6,3) on path 3-4-5-6

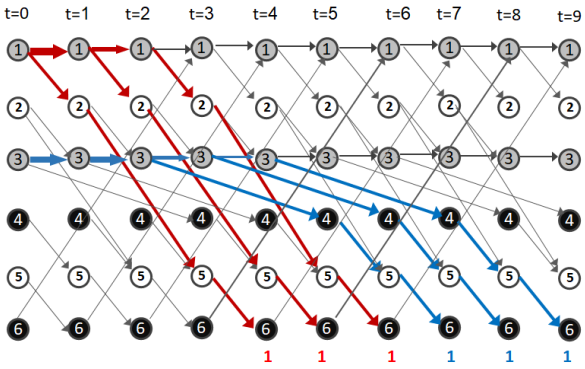


Figure 5: Routing $c_1(1,6,3)$ on path 1-2-5-6 and $c_2(3,6,3)$ on path 3-4-5-6 non-simultaneously

The multi-path-based formulation A feasible solution for given commodity flow in the single-path-based formulation corresponds to a temporary repetition of the path, where the number of repetitions depends on commodity flow demand. Figure 6 demonstrates how the makespan can increase markedly in the single-path-based formulation when the flow demand of a commodity increases. This clearly hinders the overall goal of SDD. Figure 7 showcases how the makespan can be shrunk from 13 to 12 (8%) if commodity flow items are not restricted to using only a single, unique path.

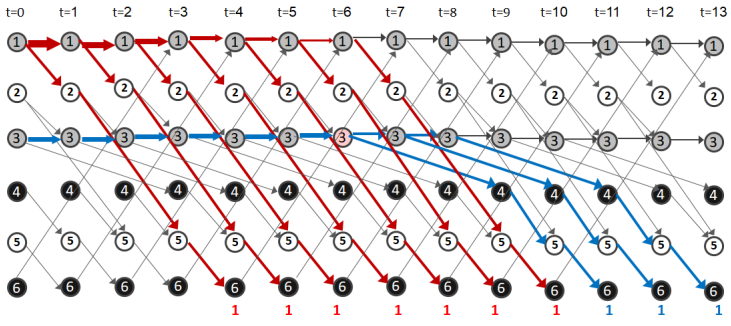


Figure 6: Routing $c_1(1,6,7)$ on path 1-2-5-6 and $c_2(3,6,3)$ on path 3-4-5-6

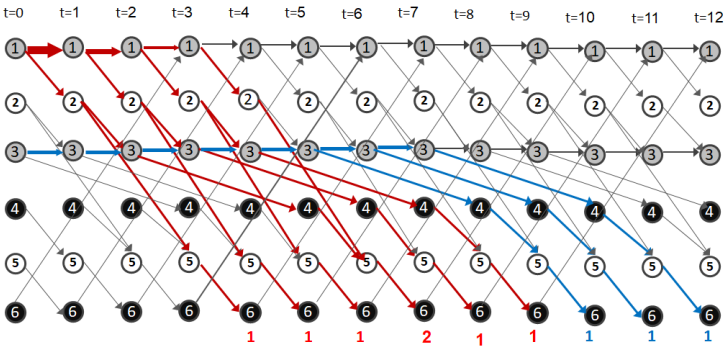


Figure 7: Improved result of Figure 6 by splitting the c_1(1,6,7) flow across 1-2-5-6 and 1-2-3-4-5-6 paths

In the multi-path-based formulation, which is based on a multi-commodity QPP formulation, the aim is to find a set of at most k_c^{max} paths through which commodity c's flow can be transshipped. Table 2 provides the list of constraints for the multi-path-based formulation.

Table 2: The multi-path-based formulation constraints

Constraint	Description
C1	Flow variables which must be coupled with their associated path-related variables
C2	The maximum number of different paths for each commodity
C3	Arc capacity constraints
C4	Path capacity constraints
C5	Flow demand constraints

A Local search-based matheuristic approach The multi-path-based formulation demands the enumeration of all commodity paths. Since this enumeration in the time-expanded graph is challenging, solving the multi-path-based formulation is also a difficult task. These characteristics motivated the design of an efficient matheuristic capable of accommodating the immense amount of data associated with conveyor operations. A local search-based algorithm is proposed to solve this formulation whose pseudo-code provided in Algorithm 1.

Algorithm 1 has two main steps: initialization and improvement. In the initialization step, a random set of paths for each commodity is selected and their exploration is realized by solving the related multi-path-based formulation. This involves calling the Gurobi MILP solver, thus providing a local optimum. The improvement step then randomly selects the first k_{max}^c paths from the feasible path list for each commodity c. The Gurobi MILP is then called, which will return a solution with optimum makespan. This improved makespan prunes the path list to the paths whose transshipment times less than or equal to the best solution obtained so far.

Algorithm 1 The local search-based algorithm

Input: I_{max} ▷ maximum number of non-improving iterations
Input: k_c^{max} ▷ maximum number of paths for commodity c
Input: tl ▷ computation time limit
Input: T ▷ time horizon
Input: $\mathcal{P}_c$ ▷ set of paths for commodity c
Output: (x,val(x)) ▷ the best commodity flow and its makespan

Initialization
$x \leftarrow 0$
$val(x) \leftarrow T$
$UB_{makespan} \leftarrow T$
$j \leftarrow 0$, $n \leftarrow 1$
for $c \in \mathbb{C}$ **do**
  $P_0 \leftarrow \cup_c p_c$ ▷ p_c randomly selects k_c^{max} paths for commodity c from $\mathcal{P}_c$
end for
$(x_0, val(x_0)) \leftarrow solveMILP(P_0, val(x), tl)$
$UB_{makespan} \leftarrow val(x_0)$
Improvement
while $n \leq I_{max}$ **do**
  for $path \in \mathcal{P}_c$ **do**
    if $path$ termination time $\geq UB_{makespan}$ **then**
      remove path from $\mathcal{P}_c$
    end if
  end for
  $P_j \leftarrow \cup_c p_c$ ▷ p_c randomly selects k_c^{max} paths for commodity c from $\mathcal{P}_c$
  $(x_{j+1}, val(x_{j+1})) \leftarrow solveMILP(P_j, val(x_j), tl)$
  if $val(x_{j+1}) \leq UB_{makespan}$ **then**
    $UB_{makespan} \leftarrow val(x_{j+1})$
  else
    $n \leftarrow n + 1$
  end if
end while

4 Computational experiments

Computational experiments have been conducted to evaluate both the single-path-based formulation and the heuristic proposed for the multi-path-based formulation. To conduct these experiments, a set of benchmark instances generated by Melchiori and Sgalambro (2015) is considered. These benchmark instance graphs have two different sizes: $s1$ (50 nodes, 186 arcs) and $s2$ (100 nodes, 292 arcs). There are 5 instances with different number of commodities $c = \{1, 2, 3, 4, 5\}$ associated with each of these sets. For each combination, two demand levels are considered where the flow demands of *level b* is double that of *level a*. This results in a total set of 20 instances. For the multi-path-based formulation, the proposed heuristic is employed to solve each instance when $k_c^{max} = \{1, 2, 3, 4, 5\}$, with the results averaged over all instances. The time horizon for performing the transshipment is defined as 16 time intervals. For both graphs it is assumed that $c_{ij} = 1$, $\lambda_{ij} = 1$, with a Gurobi time limit of 1h.

4.1 Experimental results

Tables 3 and 4 report the computational results for levels a and b, respectively. Columns $|V|$ (number of nodes), $|A|$ (number of arcs), $|c|$ (number of commodities)

denote the structure of the network. LB^* corresponds to the makespan when a complete enumeration of all the available paths for each commodity is possible and the resulting dimension of the MILPs can be handled by Gurobi. The multi-path-based formulation results are averaged over all values for k_c^{max}. $Gap_S(\%)$ and $Gap_M(\%)$ provide the makespan gaps of the single-path and multi-path-based formulations to the LB^*, respectively.

Table 3: Results for the *level a* instances

Instances size	$\|V\|$	$\|A\|$	$\|c\|$	LB^*	Single-path-based makespan	$Gap_S(\%)$	Multi-path-based makespan	$Gap_M(\%)$
s1	50	186	1	6	8	25.0	6.2	3.2
s1	50	186	2	5	8	37.5	5.2	3.8
s1	50	186	3	6	9	33.3	7.2	16.7
s1	50	186	4	5	8	37.5	7.0	28.6
s1	50	186	5	6	9	33.3	7.4	18.9
s2	100	292	1	6	9	33.3	7.0	14.3
s2	100	292	2	6	10	40.0	7.2	16.7
s2	100	292	3	6	7	16.7	7.0	16.7
s2	100	292	4	6	9	33.3	8.0	25.0
s2	100	292	5	7	tl	–	8.0	37.5

Table 4: Results for the *level b* instances

Instances size	$\|V\|$	$\|A\|$	$\|c\|$	LB^*	Single-path-based makespan	$Gap_S(\%)$	Multi-path-based makespan	$Gap_M(\%)$
s1	50	186	1	6	10	40.0	6.4	6.25
s1	50	186	2	6	10	40.0	6.2	3.22
s1	50	186	3	6	11	45.4	7.4	18.90
s1	50	186	4	6	9	33.3	7.0	14.30
s1	50	186	5	6	10	40.0	7.8	23.07
s2	100	292	1	6	11	45.4	7.0	14.30
s2	100	292	2	6	12	50.0	7.6	21.05
s2	100	292	3	6	8	25.0	7.8	23.07
s2	100	292	4	6	16	62.5	10.2	41.17
s2	100	292	5	8	tl	–	10.8	25.92

These computational results show that the single-path-based formulation is able to solve all but one instance to optimality within the time limit for both level *a* and *b*. Furthermore, these results show that the average makespan of multi-path-based formulation is far shorter than the single-path-based makespan for all instances. For both formulations, $Gap_S(\%)$ and $Gap_M(\%)$ increase as the flow demand increases. Moreover, these results show that the proposed local search-based matheuristic outperforms the single-path-based approach thereby providing evidence on how controlling the number of used paths enables flow optimization in real-world conveyor problems and capable of fulfilling the goal of SDD.

5 Conclusions

Since the time horizon available to service all same-day delivery requests is primarily devoted to transportation, only a very narrow time window remains for distri-

bution centres to sort and pack items. This study attempts to minimize the total period it takes a distribution centre conveyor to transship items from storage to the packing areas. Conveyor operation is therefore modeled as a problem which demands the routing and scheduling of multi-commodity flow. This flow is routed and scheduled by employing single-path-based and multi-path-based formulations which respect a range of real-world-based operational constraints.

Due to the complexity of solving the multi-path-based formulation, a heuristic approach was also proposed which helps to provide high quality solutions within reasonable computational time. A set of benchmark instances are used to conduct the experiments. This set includes two network sizes with various numbers of commodities and two demand levels.

Computational experiments demonstrate a proof-of-concept of our proposed formulations and evaluate the quality of the local-search-based matheuristic. Furthermore, results confirm the suitability and the superiority of solution quality of both formulations. An important direction for future work is to collect real-world-based conveyor instances. Another promising direction is to improve the proposed heuristics with a view to generating higher quality solutions for instances of the real-world conveyor.

Acknowledgements

Wim Vancroonenburg is a postdoctoral research fellow at Research Foundation Flanders - FWO Vlaanderen. This research is also funded by the FWO as part of the Strategic Basic Research (SBO) program under the ORDinL (Operational Research and Data science in Logistics) project. Editorial consultation provided by Luke Connolly (KU Leuven).

References

Banerjee, P., Banerjee, A., and Tesic, R. (1999). Optimization-based planning heuristic for material flow congestion avoidance in conveyor network design. *Production Planning & Control*, 10(2):181–193.

Boysen, N., Briskorn, D., Fedtke, S., and Schmickerath, M. (2018a). Automated sortation conveyors: A survey from an operational research perspective. *European Journal of Operational Research*.

Boysen, N., Fedtke, S., and Weidinger, F. (2018b). Optimizing automated sorting in warehouses: The minimum order spread sequencing problem. *European Journal of Operational Research*, 270(1):386–400.

Clímaco, J. C. N., Pascoal, M. M. B., Craveirinha, J. M. F., and Captivo, M. E. V. (2007). Internet packet routing: Application of a k-quickest path algorithm. *European Journal of Operational Research*, 181(3):1045–1054.

Ford, J. L. R. and Fulkerson, D. R. (1958). Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433.

Jarrah, A. I., Qi, X., and Bard, J. F. (2014). The destination-loader-door assignment problem for automated package sorting centers. *Transportation Science*, 50(4):1314–1336.

Melchiori, A. and Sgalambro, A. (2015). Optimizing emergency transportation through multicommodity quickest paths. *Transportation Research Procedia*, 10:756–765.

Melchiori, A. and Sgalambro, A. (2018). A matheuristic approach for the quickest multicommodity k-splittable flow problem. *Computers & Operations Research*, 92:111–129.

Pascoal, M., Captivo, M., and Clímaco, J. (2007). Computational experiments with a lazy version of a k quickest simple path ranking algorithm. *Top*, 15(2):372–382.

Pascoal, M. M., Captivo, M. E. V., and Clímaco, J. C. (2006). A comprehensive survey on the quickest path problem. *Annals of Operations Research*, 147(1):5–21.

Russell, M. L. and Meller, R. D. (2003). Cost and throughput modeling of manual and automated order fulfillment systems. *IIE Transactions*, 35(7):589–603.

Wahba, P. (2015). Amazon waives same-day delivery fees for prime members. *URL http://fortune. com/2015/05/28/amazon-same-day-delivery-prime/. [Online; accessed. 27-August-2015]*.

MISTA 2019

Holistic Workforce Planning: Integrating Staffing, Shift Design and Scheduling using Evolutionary Bilevel Optimization

Julian Schulte • Volker Nissen

Abstract Personnel scheduling is an important and challenging task that arises in a variety of application areas. Scheduling quality, however, strongly depends on the results of the preceding activities of staffing and shift design. This paper presents a novel approach that integrates staffing, shift design and scheduling into one optimization problem for the first time, providing a holistic view of the workforce planning process. By applying an evolutionary multi-objective bilevel algorithm, the three objectives staffing costs, number of shifts and scheduling quality are optimized simultaneously. The experimental results show that the presented holistic approach is suitable to support decision making regarding staffing as well as shift design by showing the decision maker tradeoffs between different solution alternatives. Furthermore, applying the holistic approach reveals new promising solution areas and valuable individual solutions, which would not have been found when optimizing sequentially.

1 Introduction

Getting the right people to the right place at the right time is a challenging task of high practical relevance, arising in a variety of industries such as service, transportation or manufacturing. Therefore, the problem of personnel scheduling has received significant attention in research [10, 24]. However, the activity of scheduling is only the last step in a series of preceding workforce management tasks, especially staffing and shift design [13, 23, 25].

Based on workload forecasting, the purpose of staffing is to determine the size and structure (e.g. skill-mix and contract types) of a company's workforce, i.e. the adequate future number of employees required in different categories. Since this is not an easy task even with just one type of employee, it becomes even more difficult with a heterogeneous workforce due to different qualifications or contracts [3]. Because staffing decisions have direct impact on scheduling quality, integrated planning approaches have been developed (see [16] for an overview on integrated staffing and scheduling problems and solution methods). Considering planning horizons from one day up to one year, these integrated problems are mainly solved by iteratively

Julian Schulte
Technische Universität Ilmenau
E-mail: julian.schulte@tu-ilmenau.de

Volker Nissen
Technische Universität Ilmenau
E-mail: volker.nissen@tu-ilmenau.de

alternating between the staffing part and the scheduling part of the problem as they are creating and evaluating personnel schedules based on certain staffing decisions [1, 12, 16].

Shift design in general aims at finding a suitable set of working shifts, with a shift being defined by its start time and length [13]. The problem that has received most attention in the literature is the minimum shift design problem [8, 9]. Here, the objective is to find a minimal number of shifts and to determine the number of employees needed per shift while minimizing over- and understaffing. The minimization of shifts is argued by the fact that fewer shifts result in schedules that are easier to read and manage (see [9] for an overview on shift design and related problems).

The work on shift design shows that finding a suitable set of shifts is far from trivial and has high impact on the subsequent scheduling task. Within the problem of integrated staffing and scheduling, however, the shift setting is already predefined. This paper aims at closing this gap by proposing a novel approach for integrating all three stages of the workforce management process: staffing, shift design, and scheduling. Using an evolutionary multi-objective bilevel algorithm, the three objectives staffing costs, number of shifts as well as scheduling quality are optimized simultaneously. To the best of our knowledge, there is no such approach in the literature combining all three workforce planning tasks. Furthermore, evolutionary bilevel optimization is applied to shift design (and related problems) for the first time, showing a new possibility of solving this type of problems.

The remainder of the paper is structured as follows: In Section 2 the bilevel problem of simultaneous staffing, shift design and scheduling is presented. The applied algorithms are described in Section 3. To demonstrate the advantages of the proposed holistic approach, in Section 4 the computational results of the considered case study will be discussed. Finally, the conclusions and suggestions for further research are presented in Section 5.

2 Problem Description

The following problem is based on the strategic workforce planning problem in the context of a midsized inbound call center of a German utility presented in [16]. The problem introduced in this section involves staffing, shift design as well as scheduling of the company's workforce.

2.1 Bilevel Problem

Bilevel optimization in general can be considered as a form of hierarchical optimization problem, whose hierarchical relationship is closely related to the problem of Stackelberg [20]. Here, a follower (lower-level optimization problem) optimizes his objective based on the parameters determined by the leader (upper-level optimization problem). The leader, in turn, optimizes his own objective under consideration of the follower's possible reactions [5]. Bilevel optimization problems are proven to be NP-hard [11] and can generally be formulated as follows [5, 19, 22]:

$$\min_{x \in X, y \in Y} F(x, y) \tag{1a}$$

$$\text{s.t.} \quad G(x, y) \leq 0 \tag{1b}$$

$$y \in \operatorname*{argmin}_{y \in Y} \{ f(x, y) : g(x, y) \leq 0 \} \tag{1c}$$

Here, x and y are the vectors of decision variables determined by the upper- and lower-level problem, respectively. Besides, $F(x, y)$ and $f(x, y)$ are the objective functions and G(x, y) and $g(x, y)$ the constraints of the upper- and lower-level problem.

For the considered holistic workforce planning problem, x represents the planning decision regarding staffing and shift design. For each x, the scheduling problem $f(x, y)$ will be optimized yielding personnel schedules for the considered planning horizon (represented by y). Therefore, the planning objective function $F(x, y)$ is dependent on the costs due to the staffing decision,

the number of implemented shifts as well as the quality of the created schedules at the lower level, which in turn is influenced by the determined workforce structure and the implemented shifts.

2.2 Staffing and Shift Design Problem

The call center considered in this problem has a need of k different skill types $s \in S$, with $S = \{s_1, s_2, \dots, s_k\}$. The skills are meant to be categorical, i.e. they determine the tasks each employee can perform. However, it is possible to cross-train employees so they can perform more than one type of task [3]. The qualification of an employee can therefore be seen as set of different skill combinations $q \subseteq S$. The contract type $t \in T$ of an employee determines his average weekly working time.

Within its staffing decision, the company has to predefine feasible employee types $\bar{E}$. Each employee type $\bar{e}_{qt} \in \bar{E}$ is defined by its qualification q and contract type t. Moreover, each employee type $\bar{e}_{qt}$ is linked to costs $c_{\bar{e}qt}$ that arise for employing one employee of this type over the considered planning horizon. The number of employees of each type is represented by the decision variable $x_{\bar{e}qt}$. Regarding the shift design decision, the company furthermore has to predefine a set of feasible shifts. Each shift pattern $\bar{m} \in \bar{M}$ is constrained by the operating times of the call center and the minimum and maximum shift length. Whether a shift pattern is implemented or not is represented by the decision variable $x_{\bar{m}}$ (2c). The specific problem setting will be described in Section 4.1.

The objective of the upper-level problem is to minimize the overall staffing costs as well as the number of implemented shifts (2a) subject to constraints (2b) - (2c) and the optimized scheduling decision y at the lower-level problem (2d). The decision vector passed to the lower-level is defined as $x = (x_{\bar{e}qt}, x_{\bar{m}})$.

Parameters

S	set of skills (index s)
q	qualification of an employee ($q \subseteq S$)
T	set of contract types (index t)
$\bar{E}$	set of employee types (index $\bar{e}_{qt}$)
$c_{\bar{e}qt}$	costs for an employee of type $\bar{e}_{qt}$
$\bar{M}$	set of feasible shifts (index $\bar{m}$)

Decision variables

$x_{\bar{e}qt}$	number of employees of type $\bar{e}_{qt}$
$x_{\bar{m}}$	implemented shifts

Upper-level problem

$$\min_{x \in X, y \in Y} F\left(\sum_{\bar{e}_{qt} \in \bar{E}} x_{\bar{e}qt}\, c_{\bar{e}qt}\,, \sum_{\bar{m} \in \bar{M}} x_{\bar{m}}\,, y \right) \tag{2a}$$

s.t. $$x_{\bar{e}qt} \geq 0 \; and \; integer \quad \forall\, \bar{e}_{qt} \in \bar{E} \tag{2b}$$

$$x_{\bar{m}} \in \{0, 1\} \quad \forall\, \bar{m} \in \bar{M} \tag{2c}$$

$$y \in \underset{y \in Y}{\operatorname{argmin}} \{(3a) - (3j)\} \tag{2d}$$

2.3 Scheduling Problem

The scheduling problem presented in this paper considers the daily staff scheduling of a call center over a planning horizon $W = \{1,2,\dots, w_{max}\}$. Each week of the planning horizon $w \in W$ is partitioned into periods $p \in P = \{1, 2, \dots, p_{max}\}$, representing the operating days of the call center. Moreover, each operating day again is segmented into time intervals $i \in I = \{i_1, i_2, \dots, i_{max}\}$. The specific scheduling problem setting will be described in Section 4.1.

The set of shift patterns M and set employees E are determined by the staffing and shift design decision at the upper-level, with a concrete employee for each $x_{\bar{e}qt}$ and with b_{mi} determining whether a shift pattern is covering a specific time interval. In addition, variable n_{es} determines if an employee's qualification contains skill s. The assignment of an employee $e \in E$ to a shift m on day p in week w with skill s is controlled by using the binary decision variable y^{es}_{mpw}. An employee can only be assigned to one shift each day (3e) and only if he is available and has the required skill (3c-3d). While employees may have multiple skills, each employee can only contribute one skill per shift (3e).

For each time interval i on day p in week w and each skill s a certain staffing level d^s_{ipw} has to be satisfied. The number of planned employees of each skill at time interval i is determined by variable e^s_{ipw} (3f). If a deviation $|e^s_{ipw} - d^s_{ipw}|$ arises from the staffing target, penalty points are generated by the function P_d (3g). An additional penalty is added if no employees are planned but required or vice versa.

To compensate overtime and minus hours, each employee has a flextime account u_{ew}, which is updated on a weekly basis. Therefore, the deviation of the employee's actual working time l_{ew} (3h) and contractually agreed weekly working time h_e is added to his flextime account (3i). However, to provide an equal workload distribution and to ensure that employees are staffed according to their contract types, the penalty function P_u generates penalty points based on how far employees exceeded or fell below their contractually agreed weekly working time u_{ew}/h_e. The weekly penalty is calculated by multiplying the absolute flextime value times the percentage of deviation (3j).

The objective here is to minimize the overall penalty points over the considered planning horizon (3a) subject to given constraints (3b) – (3j) described above.

Parameters

W	set of weeks in planning horizon (index w)
P	set of periods in planning week w (index p)
I	set of time intervals in planning period p (index i)
E	set of employees (index e) determined by the upper-level decision variable $x_{\bar{e}qt}$
S	set of skills (index s)
M	set of shift patterns (index m)
b_{mi}	1 if shift m covering time interval i, 0 otherwise
n_{es}	1 if employee's qualification contains skill s, 0 otherwise
a^e_{pw}	1 if employee e is available on day p in week w, 0 otherwise
d^s_{ipw}	demand of skill s at time interval i on day p in week w
e^s_{ipw}	number of planned employees with skill s at time interval i on day p in week w
P_d	demand penalty function
u_{ew}	flextime account of employee e in week w
l_{ew}	actual working time of employee e in week w
h_e	contractually agreed weekly working time of employee e
P_u	working time penalty function

Decision variable

y_{mpw}^{es} 1 if employee e is assigned to a shift m on day p in week w with skill s, 0 otherwise

Lower-level problem

$$\min_{y \in Y} f(x, P_d + P_u) \tag{3a}$$

s.t. $$a_{pw}^{e}, n_{es}, b_{mi}, y_{mpw}^{es} \in \{0,1\} \quad \forall e \in E, s \in S, m \in M, p \in P, w \in W \tag{3b}$$

$$y_{mpw}^{es} \leq a_{pw}^{e} \quad \forall e \in E, s \in S, m \in M, p \in P, w \in W \tag{3c}$$

$$y_{mpw}^{es} \leq n_{es} \quad \forall e \in E, s \in S, m \in M, p \in P, w \in W \tag{3d}$$

$$\sum_{s \in S} \sum_{m \in M} y_{mpw}^{es} \leq 1 \quad \forall e \in E, p \in P, w \in W \tag{3e}$$

$$e_{ipw}^{s} = \sum_{m \in M} \sum_{e \in E} y_{mpw}^{es} \, b_{mi} \quad \forall s \in S, i \in I, p \in P, w \in W \tag{3f}$$

$$P_d = \sum_{s \in S} \sum_{i \in I} \sum_{p \in P} \sum_{w \in W} \left| d_{ipw}^{s} - e_{ipw}^{s} \right| * \gamma_d, \qquad with$$

$$\gamma_d = \begin{cases} 500, & e_{ipw}^{s} > 0 \; and \; d_{ipw}^{s} = 0 \\ 500, & d_{ipw}^{s} > 0 \; and \; e_{ipw}^{s} = 0 \\ 1, & otherwise \end{cases} \tag{3g}$$

$$l_{ew} = \sum_{m \in M} \sum_{s \in S} \sum_{i \in I} \sum_{p \in P} y_{mpw}^{es} \, b_{mi} \quad \forall e \in E, w \in W \tag{3h}$$

$$u_{ew} = u_{e(w-1)} + (l_{ew} - h_{ew}), \qquad \text{with } u_{e0} = 0 \quad \forall e \in E, w \in W \tag{3i}$$

$$P_u = \sum_{e \in E} \sum_{w \in W} \frac{{u_{ew}}^2}{h_e} \tag{3j}$$

3 Evolutionary Bilevel Algorithm

Evolutionary Algorithms (EA) are a common metaheuristic approach to compute good solutions in an acceptable amount of time, especially when working with real world problems that otherwise cannot be solved to optimality within reasonable computation time [14]. This also applies to workforce planning and scheduling problems, with Genetic Algorithms (GA) as the most often used class of metaheuristics in this domain [3, 24]. Due to its widespread usage and successful application to similar problems, GA were chosen in this case, both to solve the upper-level staffing and shift design as well as the lower-level scheduling problem (see [15, 18, 21] for more detailed information on metaheuristic optimization in general and GA in particular).

Evolutionary bilevel optimization has been successfully applied in various areas such as economics, transportation, engineering and management, especially in strategic or long-term contexts (see [19] for a comprehensive review).

3.1 Upper-Level Algorithm

According to the taxonomy given by Talbi [22], the algorithm used to solve the presented problem of holistic workforce planning can be defined as a nested constructing approach with metaheuristics on both levels. In this type of bilevel model, an upper-level metaheuristic calls a lower-level metaheuristic during its fitness assessment. In doing so, the upper-level heuristic determines the decision vector x (here the number of employees of each type and the set of implemented shift patterns) as input for the lower-level algorithm, which in turn determines the decision vector y (optimized schedules). Both decision vectors are subsequently used to solve the bilevel problem at the upper-level. An overview of the applied bilevel algorithm is shown in Algorithm 1.

Algorithm 1 Overview of nested bilevel GA in pseudocode

popsize ← desired population size
generations ← number of generations to be evaluated
P ← build initial population of random individuals with size popsize
Best ← { }
for generations times **do**
    for each individual $P_i \in P$ **do**
        $CalculateCosts(P_i)$
        $SumShifts(P_i)$
        call lower-level GA with P_i as input
    end for
    $Best \leftarrow Fitness(P)$
    $P \leftarrow Reproduction(P, Best)$
end for
return Best

Within the here discussed problem, the three objectives staffing costs, number of shifts and scheduling penalty have to be minimized. However, both upper-level objectives are in conflict with the lower-level objective, as for example hiring multi-skilled, flexible part-time employees will yield high quality schedules but also increase the staffing costs and on the other hand, reducing the number of employees will reduce labor costs but also the scheduling quality. Furthermore, implementing a large number of shifts may increase flexibility and therefore the quality of the scheduling results.

The resulting multi-objective problem can be solved by using the concept of Pareto efficiency, which will yield a set of Pareto optimal solutions (approximated Pareto front, further referred as Pareto front). The final solution to be selected will therefore be a tradeoff among the three considered objectives staffing costs, shift count and scheduling quality (see [2, 4, 18] for more detailed information on multi-objective optimization).

To solve the upper-level multi-objective problem of minimizing staffing costs, shift count and scheduling penalty, the NSGA-II [6] is used. The chromosomes of the upper-level individuals are composed of two one-dimensional vectors, each corresponding to one subproblem. The integer encoded staffing vector $\boldsymbol{u} = (u_1, u_2, \dots, u_\varrho)$ contains $\varrho = |\bar{E}|$ values corresponding to the number of employee types. Each value u_ζ represents the number of employees to be employed for this type. However, $u_\varrho \in \mathbb{N}$ should be limited in a reasonable manner to reduce the search space. The binary encoded shift vector is denoted by $\boldsymbol{v} = (v_1, v_2, \dots, v_\vartheta)$, with $\vartheta = |\bar{M}|$. Each v_ϑ is linked to one shift pattern $\bar{m} \in \bar{M}$.

As $\boldsymbol{u}$ and $\boldsymbol{v}$ have different search spaces as well as different value ranges, they are handled independently during reproduction. However, for both vectors one-point and uniform crossover are applied, randomly selected for each reproduction process. Furthermore, intermediate crossover and random walk mutation are used for the reproduction of $\boldsymbol{u}$ (both as described in [18]). For mutation of $\boldsymbol{v}$ bit flip mutation is applied.

3.2 Lower-Level Algorithm

At the lower-level, to solve the personnel scheduling problem, an elitist GA with the same elitism principle as NSGA-II is used. Each individual is encoded as $(|E| \times (p_{max} * w_{max}))$-matrix, representing the schedules of the considered planning horizon. Each row corresponds to one concrete employee with each integer value being linked to one of the shift patterns selected at the upper-level. For reproduction, one-point and uniform crossover as well as bit flip mutation are used.

4 Experimental Study and Discussion

In this section, to demonstrate the usefulness of the proposed holistic approach, the tasks of staffing and shift design will be carried out first sequentially and then simultaneously.

4.1 Problem and Algorithm Setting

Within the investigated scenario, the call center is planning its workforce for a 4 week planning horizon $W = \{1,2,\ldots,4\}$. Furthermore, the operating days are Monday till Friday $P = \{1,2,\ldots,5\}$ from 8 a.m. to 6 p.m. $I = \{1,2,\ldots,10\}$ (due to the strategic context, hourly scheduling intervals were chosen). Shifts are allowed in the range between 4 and 8 hours, resulting in 25 possible shift patterns. Regarding its staffing decision, the company has a need of two different skills $S = \{\text{agent}, \text{support}\}$. The forecasted demand of agents is highly volatile both during the day and across the planning horizon. The demand of support employees is calculated based on a staffing ratio of one support for each four agents. Table 1 shows the predetermined employee types with related costs. The costs of each employee type are represented by a relative factor summing up annual wages, payroll taxes, overhead and training costs.

Table 1 Setting of the staffing problem

Contract type	Qualification	Costs
20 h / 40 h	agent	0.6 / 1
20 h / 40 h	support	0.65 / 1.1
20 h / 40 h	agent - support	0.75 / 1.3

For the upper-level GA, a population size of 60, a generation number of 100 and n=10 restarts were chosen, with each restart having a random initial population. Since all results on the upper level are evaluated according to the same algorithm, the goal on the lower level is not to find the best overall solution, but to achieve stable results in order to avoid outliers. Therefore, considering the tradeoff regarding computation time, solution quality and solution noise, the lower-level GA was configured with a population size of 50 and a generation number of 200.

On both levels, the mutation rate was set to 1/v, with v being the number of genes of the encoded individual, and the application of the available crossover operators was uniformly distributed. The fitness at the upper-level was evaluated by Eq. (2a), for the fitness evaluation at the lower-level Eq. (3a) was used.

The optimization software was written in Python/Cython and all experiments were executed on a cluster of 12 Windows 10 machines with Intel Core i7-8700 machines (6 cores, maximum clock rate 3.19 GHz) and 16 GB RAM. The evaluation of the lower-level fitness function was spread among all available cores. One restart took about 1 hour, ten restarts about 8 hours (due to the better exploitation of cores).

4.2 Sequential Workforce Planning

In this subsection, the workforce planning tasks of staffing and shift design are executed one after the other as usual. The sequential problems can be considered as modifications of the problem described in Section 2, taking into account either the staffing or shift design problem and objectives, respectively.

The sequential workforce planning study starts with the staffing task. For this, at first an initial shift setting had to be chosen. Using problem specific insights and considering the volatile demand, four shifts were selected. Two 8 hour shifts covering the base demand starting at 8 a.m. and 10 a.m. and two 4 hour shifts covering peaks in the morning and the afternoon, starting at 10 a.m. and 2 p.m.. Using this setting, the algorithm was executed while ignoring the objective of shift count, resulting in a conventional integrated staffing and scheduling problem (see [16]). As final result, the Pareto fronts of all restarts were combined and a new Pareto front was created (see Fig. 1).

Fig. 1 shows the expected behavior of higher quality schedules with increasing staffing costs, due to more flexible employees. The best possible staffing setting (with the selected shift setting) is marked as solution A. Solution B is selected as an efficient alternative, which reduces staffing costs by 13% with an increase of penalty points by only 3%. The details of both solutions are shown in Table 2. The two staffing decisions, namely the best possible and the more efficient one, will be referred to in the following as scenario A and B. The shifts are encoded as "starting hour"_"duration (h)" (24h clock).

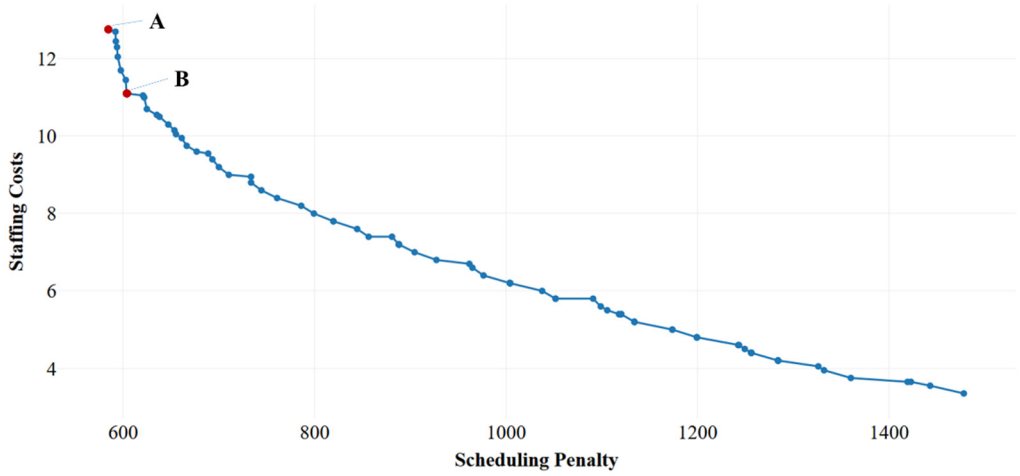


Fig. 1 Possible staffing decisions (Pareto front of n=10 optimization runs)

Table 2 Selected staffing solutions

	Costs	Penalty	#Shifts	Agent		Support		Ag.-Sup.		Shifts
				40	20	40	20	40	20	
A	12.75	585.6	4	5	5	0	3	1	2	8_8, 10_8, 10_4, 14_4
B	11.1	603.6	4	4	6	2	0	1	0	8_8, 10_8, 10_4, 14_4

For the shift design task, the algorithm was executed again with the staffing decisions of solutions A and B. Here, in contrast to the previous experiment, the staffing setting is fixed and only the objective of minimizing the number of shifts is considered. The results of both experiments are shown in Fig. 2 and Fig. 3 as well as in Table 3 and 4, respectively.

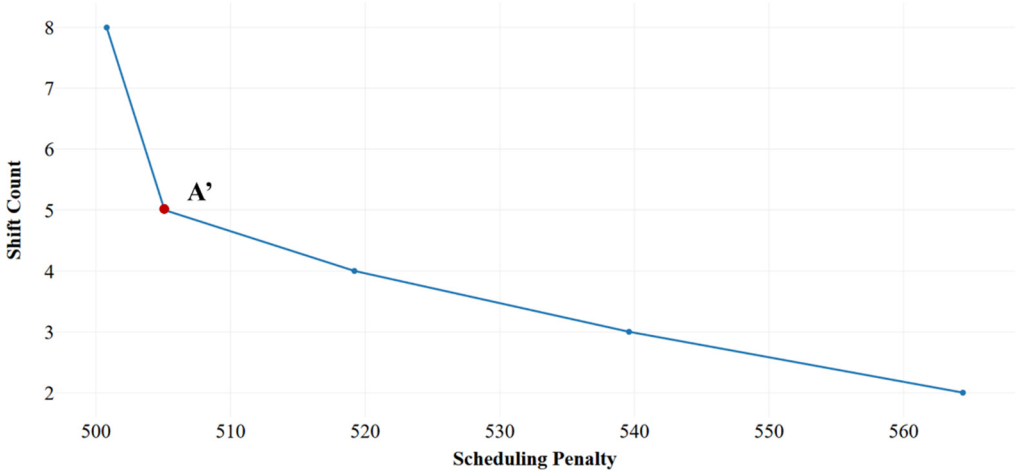


Fig. 2 Possible shift design decisions for scenario A (Pareto front of n=10 optimization runs)

Table 3 Shift design solutions for scenario A

	Costs	Penalty	#Shifts	Agent		Support		Ag.-Sup.		Shifts
				40	20	40	20	40	20	
	12.75	500.5	8	5	5	0	3	1	5	8_4, 8_5, 8_8, 9_6, 9_8, 10_8, 13_5, 14_4
A'	12.75	505.1	5	5	5	0	3	1	5	8_4, 8_7, 9_8, 10_8, 14_4
	12.75	519.2	4	5	5	0	3	1	5	8_4, 9_8, 10_8, 13_5
	12.75	539.6	3	5	5	0	3	1	5	8_4, 10_8, 13_4
	12.75	564.4	2	5	5	0	3	1	5	8_4, 10_8

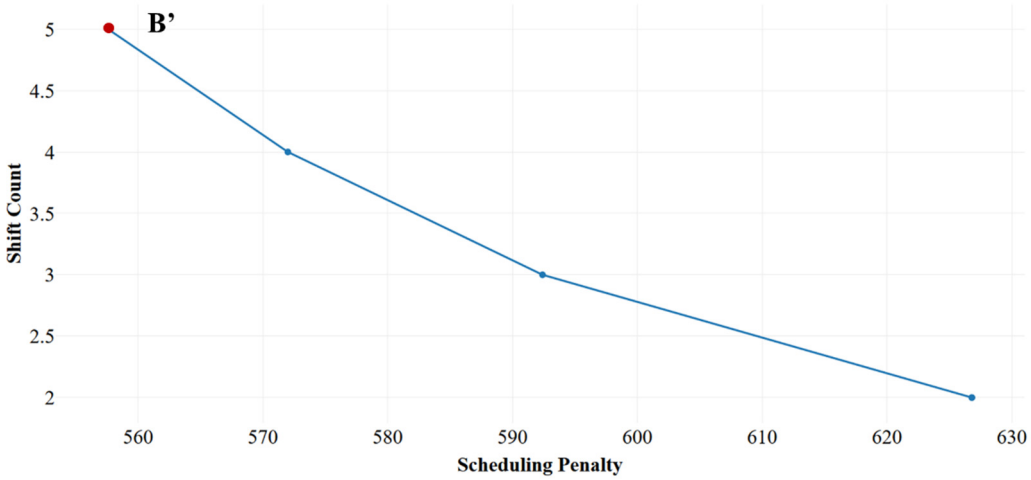


Fig. 3 Possible shift design decisions for scenario B (Pareto front of n=10 optimization runs)

Table 4 Shift design solutions for scenario B

	Costs	Penalty	#Shifts	Agent		Support		Ag.-Sup.		Shifts
				40	20	40	20	40	20	
B'	11.1	557.6	5	4	6	2	0	1	0	8_4, 8_8, 9_6, 10_8, 14_4
	11.1	572	4	4	6	2	0	1	0	8_4, 8_8, 10_8, 14_4
	11.1	592.4	3	4	6	2	0	1	0	8_4, 8_8, 10_8
	11.1	626.8	2	4	6	2	0	1	0	8_8, 10_8

Examining the shift design results of both scenarios, it becomes apparent that the scheduling quality could further be improved by selecting different shift patterns. The second solution of scenario B (see Fig. 3 & Table 4) shows that the initial shift decision (see Table 2) was already quite reasonable. However, this decision was improved by letting the short shift 10_4 (start at 10 a.m., duration 4 hours) start 2 hours earlier (8_4). Looking at the second solution of scenario A (see Fig 2 & Table 3), the scheduling quality as compared to A in Table 2 was improved by about 14% by adding only one more shift. As the 4.6 additional penalty points seem neglectable compared to the three additional shifts (in comparison to the first solution), this solution will be used in the further discussion. In the remainder of this section, the solutions of both scenarios with 5 shifts will be referred as solutions A' and B'.

It may be assumed that, due to the higher flexibility, with an increasing number of shifts the scheduling quality increases as well, which is reflected by the results of both scenarios. Nevertheless, in both scenarios the maximum number of shifts (5 and 8) is way below the maximum amount of 25 possible shifts, which at least relativizes this assumption.

4.3 Holistic Workforce Planning

In this subsection, the entire problem described in Section 2 will be solved and compared to the results of both scenarios described above. Fig. 4 shows the three-dimensional Pareto front for the objectives staffing costs, shift count, and scheduling penalty. As within the previous

sequential steps, the pattern of decreasing penalty with increasing staffing costs and number of shifts of the previous sequential steps can be found.

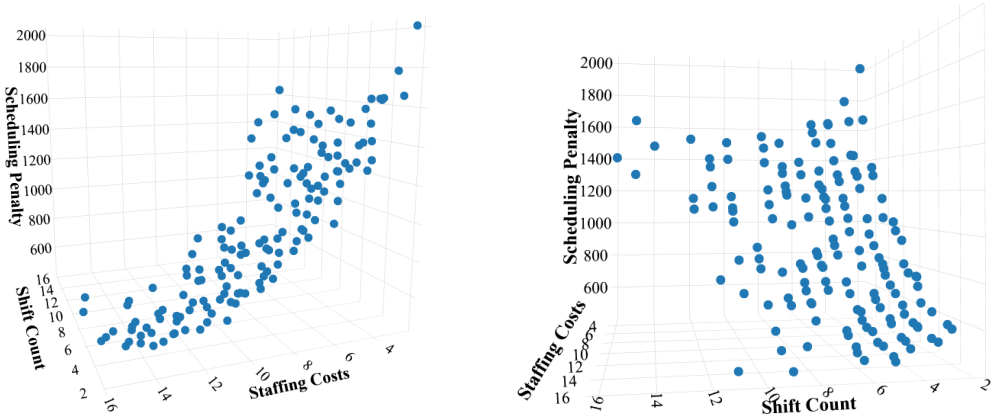


Fig. 4 Possible staffing and shift design decisions (Pareto front of n=10 optimization runs)

The most interesting part, however, is the target area below the penalty of 600 (see Fig. 5 & Table 5). While the smallest penalty found during sequential workforce planning was 500.5 (see Table 3), simultaneously optimizing staffing and shift design was able to further reduce the minimal penalty by about 10% to 452 (see Table 5, Sol. C). In general, applying the holistic workforce planning approach reveals the circled area, which would have remained undiscovered by the sequential approach (see Fig. 5). Providing a broader basis for decision making can therefore be considered as main advantage of integrating all aspects of workforce planning into one optimization problem. For further discussion, solutions C to F are exemplarily selected from this area (see Table 5).

The solutions which are closest to those selected in the previous scenarios are labeled as A'' and B'' (see Fig 5 & Table 5). When comparing these results to those from the sequential steps, it becomes apparent that latter ones are dominating those from the holistic approach. This could be explained by the fact that by combining the staffing and shift design problem the search space increases, which would have required an evaluation of a larger number of solutions to better exploit this more complex search space.

To verify the assumption of a better search space exploitation within the sequential approach, the staffing task is repeated with a shift setting found in the circled area. To keep a reasonable amount of shifts, the shift setting of solutions E and F were chosen. This decision is supported by the fact that most of the shifts can also be found in solutions C and D.

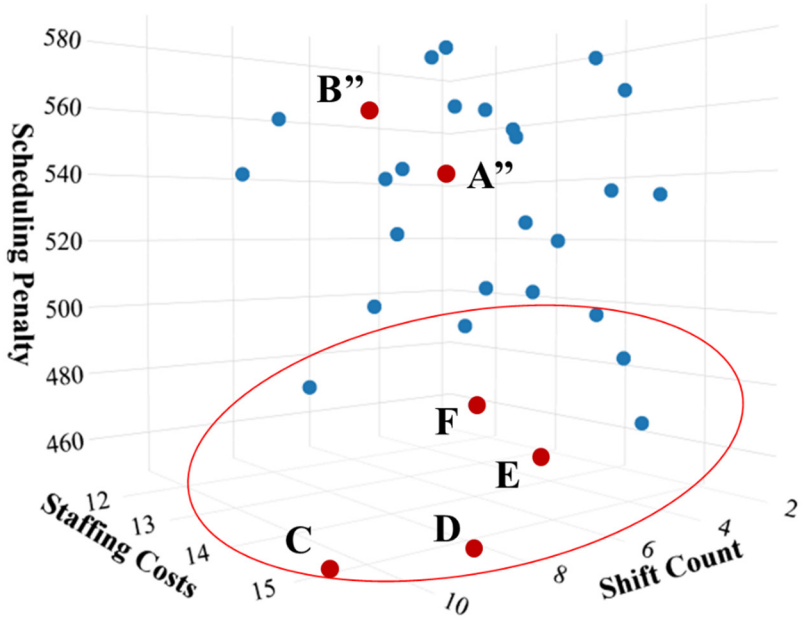


Fig. 5 Target area of holistic workforce planning solutions

Table 5 Selected holistic workforce planning solutions

	Costs	Penalty	#Shifts	Agent		Support		Ag.-Sup.		Shifts
				40	20	40	20	40	20	
A’’	13	541.2	5	2	9	0	2	1	4	8_4, 8_5, 10_5, 10_8, 14_4
B’’	11.65	564.4	5	4	5	1	2	0	3	8_4, 8_8, 9_8, 10_8, 14_4
C	15.25	452	11	1	10	0	0	0	11	8_4, 8_5, 8_8, 9_5, 10_5, 10_7, 10_8, 11_7, 12_6, 13_5, 14_4
D	15.65	453.8	9	0	6	0	3	2	10	8_4, 8_5, 9_4, 9_6, 9_8, 10_6, 10_8, 13_5, 14_4
E	14.85	463.8	6	0	12	0	2	2	5	8_4, 8_5, 9_6, 10_8, 13_5, 14_4
F	14	473.9	6	0	12	1	1	1	5	8_4, 8_5, 9_6, 10_8, 13_5, 14_4

Fig. 6 shows the results of the integrated staffing and scheduling experiment with the shift setting of solutions E and F. Here, solution A* is dominating solution A’ in respect of staffing costs and scheduling penalty. Solutions B* and B’ are almost similar, with B* having slightly lower costs and slightly higher penalty. Solution G has the smallest penalty at highest costs

across all experiments, with H being seen as an efficient alternative. Solution I dominates all solutions shown in Table 5.

Furthermore, it becomes apparent that due to the selected shift setting with only one full time shift, part time employees are favored. If one might prefer a more full time workforce, this should either be reflected in the staffing costs or a different shift setting of the holistic planning results (Fig. 5) should be chosen.

When looking at the highlighted solutions, the assumption of a better search space exploitation during the sequential experiments can be verified. The advantage of the holistic approach, however, comes to bear again in this last experiment, as without it the selected shift design decision would not have been discovered. Moreover, by adjusting the upper-level algorithm to the increased complexity (e.g. larger population and more generations) the holistic approach very likely would achieve the same search space exploitation.

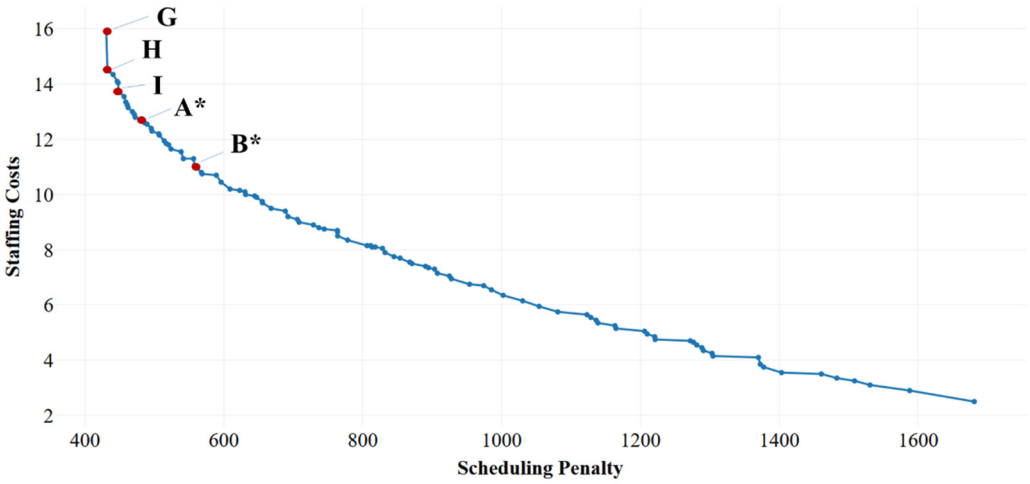


Fig. 6 Possible staffing decisions 2 (Pareto front of n=10 optimization runs)

Table 6 Possible staffing decisions 2

	Costs	Penalty	#Shifts	Agent		Support		Ag.-Sup.		Shifts
				40	20	40	20	40	20	
G	15.9	430.7	6	1	10	0	1	0	11	8_4, 8_5, 9_6, 10_8, 13_5, 14_4
H	14.5	432.4	6	1	8	0	3	0	9	8_4, 8_5, 9_6, 10_8, 13_5, 14_4
I	13.75	448.45	6	1	13	0	3	0	4	8_4, 8_5, 9_6, 10_8, 13_5, 14_4
A*	12.7	481.25	6	1	10	0	3	0	5	8_4, 8_5, 9_6, 10_8, 13_5, 14_4
B*	11.05	557.75	6	1	11	0	3	0	2	8_4, 8_5, 9_6, 10_8, 13_5, 14_4

5 Conclusion

In this paper, a novel approach was proposed for integrating three main stages of the workforce management process: staffing, shift design and scheduling. To demonstrate the advantages of the holistic approach, the tasks of staffing and shift design were first executed sequentially and afterwards simultaneously. The results indicate that when performing staffing and shift design sequentially, the staffing decision is limited by the predefined shifts. The shift design decision, on the other hand, is limited by the selected staffing setting. The proposed holistic workforce planning approach solves this dilemma and shows the decision maker the 'big picture' of the considered problem by revealing tradeoffs regarding staffing and shift design decisions, as well as new promising solution areas and valuable individual solutions, which would not have been found when optimizing sequentially.

For well-founded decision making, however, deeper insights into the problem structure are beneficial [7]. Further research should therefore focus on analyzing the decision variables of the upper-level problem, e.g. for interaction effects, as initially proposed in [17]. Possible questions here could be whether there is a set of dominant shifts, which shifts harmonize with each other or if there are shifts that work better for certain staffing settings. Furthermore, additional parameters could be integrated into the upper-level problem, such as policy making as shown in [12]. In order to achieve best possible results and to avoid an additional optimization step (see Fig. 6), further research should also focus on optimizing the settings of the upper-level algorithm.

References

1. Beliën J, Demeulemeester E, A branch-and-price approach for integrating nurse and surgery scheduling, European Journal of Operational Research 189(3), 652–668 (2008)
2. Branke J (ed.), Multiobjective optimization. Interactive and evolutionary approaches. Lecture Notes in Computer Science, vol. 5252. Springer, Berlin (2008)
3. Bruecker P de, van den Bergh J, Beliën J, Demeulemeester E, Workforce planning incorporating skills. State of the art, European Journal of Operational Research 243(1), 1–16 (2015)
4. Coello Coello CA, Lamont GB, van Veldhuizen DA, Evolutionary algorithms for solving multi-objective problems, 2nd edn. Genetic and evolutionary computation series. Springer, New York, NY (2007)
5. Colson B, Marcotte P, Savard G, An overview of bilevel optimization, Ann Oper Res 153(1), 235–256 (2007)
6. Deb K, Pratap A, Agarwal S, Meyarivan T, A fast and elitist multiobjective genetic algorithm. NSGA-II, IEEE Trans. Evol. Computat. 6(2), 182–197 (2002)
7. Deb K, Srinivasan A, Innovization. In: Cattolico, M. (ed.) Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06. the 8th annual conference, p. 1629. ACM Press, New York, New York, USA (2006)
8. Di Gaspero L, Gärtner J, Kortsarz G, Musliu N, Schaerf A, Slany W, The minimum shift design problem, Ann Oper Res 155(1), 79–105 (2007)
9. Di Gaspero L, Gärtner J, Musliu N, Schaerf A, Schafhauser W, Slany W, Automated Shift Design and Break Scheduling. In: Uyar, A.S., Ozcan, E., Urquhart, N. (eds.) Automated Scheduling and Planning: From Theory to Practice, pp. 109–127. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
10. Ernst AT, Jiang H, Krishnamoorthy M, Sier D, Staff scheduling and rostering. A review of applications, methods and models, European Journal of Operational Research 153(1), 3–27 (2004)
11. Hansen P, Jaumard B, Savard G, New Branch-and-Bound Rules for Linear Bilevel Programming, SIAM J. Sci. and Stat. Comput. 13(5), 1194–1217 (1992)
12. Maenhout B, Vanhoucke M, An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems, Omega 41(2), 485–499 (2013)

13. Musliu N, Schaerf A, Slany W, Local search for shift design, European Journal of Operational Research 153(1), 51–64 (2004)
14. Nissen V, Applications of Evolutionary Algorithms to Management Problems. In: Moutinho, L., Sokele, M. (eds.) Palgrave Handbook of Innovative Research Methods in Management. Palgrave Macmillan, Basingstoke (2017)
15. Rothlauf F, Design of Modern Heuristics. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
16. Schulte J, Günther M, Nissen V, Evolutionary Bilevel Approach for Integrated Long-Term Staffing and Scheduling. In: Proceedings of the 8th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2017), pp. 144–157, Kuala Lumpur, Malaysia (2017)
17. Schulte J, Feldkamp N, Bergmann S, Nissen V, Knowledge Discovery in Scheduling Systems Using Evolutionary Bilevel Optimization and Visual Analytics. In: Deb, K., Goodman, E., Coello Coello, C.A., Klamroth, K., Miettinen, K., Mostaghim, S., Reed, P. (eds.) Evolutionary Multi-Criterion Optimization, pp. 439–450. Springer International Publishing, Cham (2019)
18. Sean Luke, Essentials of Metaheuristics. Lulu (2013)
19. Sinha A, Malo P, Deb K, A Review on Bilevel Optimization. From Classical to Evolutionary Approaches and Applications, IEEE Trans. Evol. Computat. 22(2), 276–295 (2018)
20. Stackelberg H, The theory of market economy. Oxford University Press, Oxford (1952)
21. Talbi E-G, Metaheuristics. From design to implementation. Wiley Series on Parallel and Distributed Computing, v.74. John Wiley & Sons, Hoboken, NJ (2009)
22. Talbi E-G, A Taxonomy of Metaheuristics for Bi-level Optimization. In: Talbi, E.-G. (ed.) Metaheuristics for Bi-level Optimization, vol. 482. Studies in Computational Intelligence, pp. 1–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
23. Tien JM, Kamiyama A, On Manpower Scheduling Algorithms, SIAM Rev. 24(3), 275–287 (1982)
24. van den Bergh J, Beliën J, Bruecker P de, Demeulemeester E, Boeck L de, Personnel scheduling. A literature review, European Journal of Operational Research 226(3), 367–385 (2013)
25. Voudouris C, Lesaint D, Owusu G, Service Chain Management. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

MISTA 2019

Branch-and-price-and-cut Algorithm of Selective Pickup and Delivery Problem with Loading Cost and Time Window

XUE Li • CHE Ada

Abstract

Selective Pickup and Delivery Problem with Loading Cost and Time Window (SPDPLCTW) is a new variant of the PDP in a many-to-many commodity structure. In the SPDPLCTW, a fleet of vehicles is dispatched to serve a set of customers under the restrictions of the vehicle capacity, maximum duration, and time window. As the customer requests must be satisfied in the SPDPLCTW, each delivery is served once. Similar to the VRP, the depot is assumed as a pickup node with unlimited storage and if necessary, vehicles depart from the depot with an initial load to fulfill the deliveries. Pickup is only visited when adding the load of the pickup can save transportation cost. To solve this problem, I propose a branch-and-price-and-cut algorithm. An ad hoc label-setting algorithm with a linear structure is devised, where tabu heuristic is employed to accelerate the label-setting algorithm. The experiment instances are generated by a designed data generator, and computational results show that the proposed algorithm can handle most instances efficiently.

1 Introduction

we present a new generation of PDP: the selective load pickup and delivery problem with loading cost and time window (SPDPLCTW). The SPDPLCTW is inspired by the station model applied by B2C companies in China. Until 2015, Alibaba built more than ten thousand county stations in 25 provinces in China and has decided to increase the number to a hundred thousand in 5 years. Ririshun, an offline logistics company hold by Haier, had built more than 6000 stations in 91 cities in China. To take advantage of the cheap resident real state and shorten the distance between the storage and customers, the B2C companies build some small stations to storage the commodities,

XUE LI
School of Management, Northwestern Polytechnical University
E-mail: xueli@nwpu.edu.cn

CHE ADA
\School of Management, Northwestern Polytechnical University
E-mail: ache@nwpu.edu.cn

where vehicles can collect loads to serve more customers. Because the commodities transported in B2C business are usually homogeneous like the electronic appliances and so on, the commodity collected from pickups can be considered as undifferentiated. Another remarkable service in the B2C business is the reservation service ahead. As the commodity delivered in B2C business can be heavy appliance like refrigerators or equipment required to install, the service time are always appointed. Another good example is the internal transfer requests in retail industry. To commodity like cloth and electronic appliance, the retailers often have several stores distributed in one area. As the sales of stores are random, the retailers sometimes need to transfer the commodities from some surplus stores to sell out stores. Using this strategy, retailers can balance the inventories between stores with the consideration of the transportation cost.

In the SPDPLCTW, the loading cost is incorporated in the objective to reflect the transportation cost in the *toll-by-weight* scheme. The time window constraint is applied to reflect the reservation service. As the requests of customers must be once satisfied, each delivery must be served only once. Similar to the classical VRP, the depot is assumed as a pickup with unlimited storage and if necessary, vehicles can depart from the depot with an initial load to fulfill deliveries. The selective pickup is introduced to simulate the station, and the pickup is only visited when adding the load of the pickup can save the transportation cost.

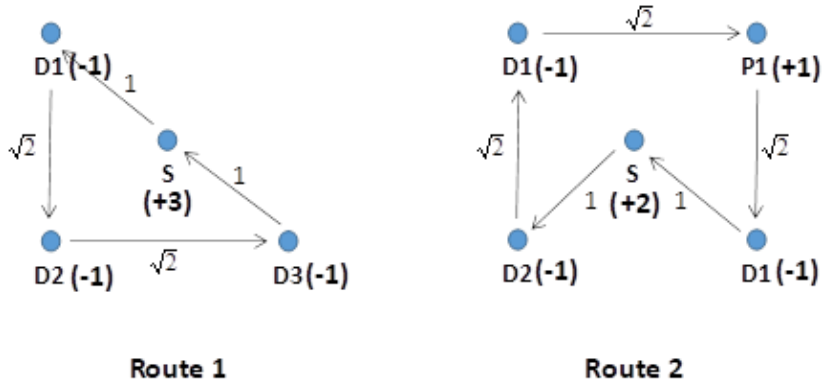


Figure 1 Example of the effect of selective pickups

The SPDPLCTW can be consider as a new generation of the vehicle routing problem with loading cost (VRPLC) with selective pickups, time windows. Despite the time window constraints in the SPDPLCTW, the effect of selective pickups can be illustrated by the example shown in figure 1. Assuming S is the depot of a vehicle with capacity of 3 units. D1, D2, D3 are deliveries and P1 is a selective pickup. The distance between nodes and the demands of each node are showed in figure 1. Set the load cost coefficient =1.0. In route 1, the vehicle departs from the depot S with 3 unit loads and visits nodes as the sequences of route 1. The distance cost of route 1 is 2+2 √2 and the loading cost is 3+3 √2. In route 2, the vehicle departs from the depot S with 2 unit loads and uploads 1 unit at P1. The distance cost of route 2 is 2+3 √2 and the loading cost 2+2 √2. Totally, the transportation cost of route 1 is 5+5 √2 which is bigger than the cost of route 2 that equals to 4+5 √2. Obviously, the cost of optimal solution deceases by introducing the selective pickups.

Theoretically, the SPDPLCTW belongs to the many-to-many structure, one commodity pickup and delivery problem. In many-to-many structure, there are several vertexes that can serve as the source or as the destination for each commodity. A fleet of vehicles transports the commodities within the vertexes until all customers' demands are satisfied. The related literature refers mostly to single vehicle scenarios and is rather limited comparing to the one-to-one structure. The Swapping Problem (SP) is a many-to- many, n-commodity PDP proposed byAnily and Hassin [1]. It deals with the swapping of objects between vertices with a single vehicle and the vehicle capacity is one unit. The preemptive and non-preemptive versions of the SP were studied separately by Wang et al. [2] and Anily et al. [3]. Anily et al. [4] studied the cases where vertices are located on a line with an exact algorithm and derived some general properties of optimal solutions.

The one-commodity pickup and delivery traveling salesman problem (1-PDTSP) is the most well-known many-to-many, one-commodity problem. It can be considered as a multiple demand and non-preemption case of the SP. In the 1-PDTSP, the set of customer locations is partitioned into pickup and delivery customers to be served by a single vehicle based at the depot with an initial load. Hernández-Pérez and Salazar-González [5] first proposed 1-PDTSP. Hernández-Pérez and Salazar-González [6] provided a branch-and-cut exact algorithm. Hernándezpérez and Salazargonzález [7] introduced two heuristic algorithms to solve 1-PDTSP and a hybrid GRASP/VND heuristic is devised by Hernández-Pérez et al. [8]. Mladenović, Urošević [9] presented a new heuristic based on general variable neighborhood search. A similar problem to the 1-PDTSP is the q-delivery traveling salesman problem (Q-DTSP), which is also a many-to-many, one-commodity PDP. It consists of uploading and delivering single objects from source to sink vertices by a single vehicle with capacity Q. Most existing researches about the Q-DTSP focus on the development of approximation algorithms[10,11].

In the 1-PDTSP, all pickups have to be visited once and the depot can provide unlimited commodities to the vehicles. The basic concept of the SPDPLCTW is principally motivated by 1-PDTSP. Despite of the loading cost, SPDPLCTW cannot be simply considered as the 1-PDTSP in multi vehicle situation. First, in the 1-PDTSP, the visit of the pickup customers is necessary. In SPDPLCTW, the visit of pickups is selective. Second, in the 1-TSPPD, the initial load in depot is predefined as a fixed number: $q_0 = -\sum_1^n q_i$. In the SPDPLCTW, the initial load of each vehicle is a flexible number depend on the pickups and the deliveries in the route, and the sum of all initial loads may be bigger than $-\sum_1^n q_i$ because the visit of pickups is selective.

Another related research is the selective pickup and delivery problem (SPDP), which is firstly introduced by Liao and Ting [11]. The SPDP holds many-to-many structure, vehicle capacity constraint and maximum duration constraint. In SPDP, all delivery must be visited once but the only the significant pickup can be visited most once. The rental bike system can be a good example for it, as rental bike is undifferentiated commodity. The management agency has to temporarily deliver bikes from some rental stations with sufficient bikes to other rental stations that are lack of bikes. Liao and Ting [12] solved the SPDP and its multi-vehicle cases by using the evolutionary algorithm and the multi-vehicle case and multi-objective case are studied with the genetic algorithm [13,14].

The SPDPLCTW can be considered as the SPDP starting from a pickup with infinite capacity and restricted the loading cost and the time window constraints. As there is no exact algorithm for the SPDP, the SLPDPLCTW is still meaningful to study.

We conduct a detail computational study on the SPDPLCTW with a branch-and-price-and-cut algorithm. A strong lower bound is provided by the linear programming (LP) relaxation of a set-partitioning model with an exponential number of decision variables. I also test several valid inequalities in literature, but only the rounded capacity inequalities are significant. The bounded bidirectional search strategy is inefficient in the SPDPLCTW due to the structural propriety. The LP relaxation of the set-partitioning modeling is solved by the column generation and the elementary shortest path problem with loading cost is solved by a label-setting algorithm. The tabu heuristic is employed to improve the efficiency of label-setting algorithm. The branch-and-price-and-cut algorithm was tested on a set of instances which were generated by a data generator derived from the data generator used in Ropke and Cordeau [15].

This paper is conducted as following: In section 2 we present the arc-flow model and the set-partitioning model for the SPDPLCTW. Next we introduce the pricing problem in column generation and the label-setting algorithm as well as tabu heuristic column generator in Section 3. The detail of the branch-and-price algorithm, including valid inequalities, initial solution generation heuristic, branch-and-bound tree management scheme and branching strategies, are presented in Section 4. The result of computational experiments and conclusion are discussed in section 5 and 6.

2 Mathematical Models

Let $G = (V, A)$ symbolize the complete graph in SPDPLCTW. $V = \{0, \ldots, n+1\}$ is the node set, where nodes 0 and $n+1$ are two duplicates of the depot, and a given demand q_i associated with customer i, where $q_i > 0$ if i is a pickup and $q_i < 0$ if i is a delivery. $[e_i, l_i]$ is the time window of customer i. $A = \{(i, j) \mid i, j \in V, i \neq j\}$ is the arc set. And each arc $(i, j) \in A$ has a travel distance $d_{i,j}$, which is assumed to satisfy the triangle inequality, i.e., $d_{i,j} \leq d_{i,k} + d_{k,j}$ for $(i, j), (i, k), (k, j) \in A$ .The pickup set P is defined as $P = \{i \in V \setminus \{0, n+1\}, q_i > 0\}$ and delivery set D is defined as $D = \{i \in V \setminus \{0, n+1\}, q_i < 0\}$. Let K denote the set of identical vehicles with the load capacity Q and the maximal duration T. A vehicle departs from the depot, visits a set of customers and back to the depot with empty load, which called a route. The vehicle departs with an initial load from the depot, which cannot exceed the capacity Q. λ is the objective coefficient to balance the distance cost and the loading cost.

2.1 Arc-flow Model

Let $x_{i,j,k}$ be a binary decision variable which is set to 1 if arc (i, j) is used by vehicle k and 0 otherwise. Let $a_{i,k}$ be non-negative decision variables representing the arrival time of vehicle k at node $i \in P \cup D$. Let $w_{i,k}$ be the accumulative load of vehicle k at node i and $w_{0,k}$ be the

initial load of vehicle k. Let W be a sufficiently large positive number to ensure corresponding constraints. The arc-flow model for the SPDPLCTW can be formulated as follows:

$$min \sum_{k \in K} \sum_{(i,j) \in A} d_{i,j} x_{i,j,k} + \lambda \times \sum_{k \in K} \sum_{(i,j) \in A} w_{i,k} \times d_{i,j} x_{i,j,k} \tag{1}$$

$$s.t. \sum_{k \in K} \sum_{(i,j) \in A} x_{i,j,k} = 1, \forall i \in D \tag{2}$$

$$\sum_{k \in K} \sum_{(i,j) \in A} x_{i,j,k} \leq 1, \forall i \in P \tag{3}$$

$$\sum_{(j,i) \in A} x_{j,i,k} = \sum_{(i,j) \in A} x_{i,j,k}, \forall i \in P \cup D, \forall\ k \in K \tag{4}$$

$$\sum_{(0,j) \in A} x_{0,j,k} = 1, \forall k \in K \tag{5}$$

$$\sum_{(i,n+1) \in A} x_{i,n+1,k} = 1, \forall k \in K \tag{6}$$

$$w_{j,k} \geq w_{i,k} + q_j + (x_{i,j,k} - 1)W, \forall (i,j) \in A, \forall k \in K \tag{7}$$

$$w_{i,k} \geq w_{j,k} - q_j + (x_{i,j,k} - 1)W, \forall (i,j) \in A, \forall k \in K \tag{8}$$

$$0 \leq w_{0,k} \leq Q, \forall k \in K \tag{9}$$

$$w_{n+1,k} = 0, \forall k \in K \tag{10}$$

$$a_{j,k} \geq a_{i,k} + d_{i,j} + (x_{i,j,k} - 1)W, \forall (i,j) \in A, \forall k \in K \tag{11}$$

$$e_i < a_{i,k} < l_i, \forall i \in P \cup D \tag{12}$$

$$a_{n+1,k} - a_{0,k} \leq T, \forall k \in K \tag{13}$$

$$max(0, q_i) \leq w_i \leq min(Q, Q + q_i), \forall i \in P \cup D \tag{14}$$

$$x_{i,j,k} \in \{0,1\}, a_{i,k} \geq 0, \forall i \in V, \forall (i,j) \in A, \forall k \in K \tag{15}$$

The objective function (1) minimizes the total cost consisting of the distance cost and the loading cost. The constraint (2) ensures that every delivery must be visited only once and constraint (3) guarantees that every pickup can be mostly visited once. Following constraint (7) and (8), the vehicle will upload and unload the commodity of nodes along with the route sequence. Constraint (9) and (10) guarantee the initial load of vehicle is less than the vehicle capacity and each vehicle back to the depot with empty load. Constraint (12) guarantee the arrival time of customer i within its time window $[e_i, l_i]$. Constraint (13) and (14) are imposed for vehicle capacity constraints and maximum duration constraints.

2.2 Set-partitioning Model

Let Ω denote the set of feasible routes, which satisfy the time window constraints, the capacity constraint and the maximal duration constraint. Let c_r denote the cost of route r and $\alpha_{i,r}$ be a binary number which is equal to 1 if route r covers customer i and 0 otherwise. Binary decision variable θ_r is set to 1 if route r is selected in the optimal solution and 0 otherwise. The set-partitioning model of the SPDPLCTW can be formulated as follows:

$$\min \sum_{r\in\Omega} c_r\theta_r \tag{16}$$

$$s.t. \sum_{r\in\Omega} \alpha_{i,r}\theta_r = 1, \forall i \in D \tag{17}$$

$$\sum_{r\in\Omega} \alpha_{i,r}\theta_r \le 1, \forall i \in P \tag{18}$$

$$\sum_{r\in\Omega} \theta_r \le |K| \tag{19}$$

$$\theta_r \in \{0,1\}, \forall r \in \Omega \tag{20}$$

Constraint (17) ensures that each delivery must be served exactly once. Constraint (18) ensures that the pickup is selective and can be at most visited once. By relaxing the constraint (20) with $\theta_r \ge 0, \forall r \in \Omega$, a linear relaxation of the master problem (LMP) can be got, whose optimal solution is the lower bound of the original problem. The LMP problem cannot be directly solved by a standard LP solver, but can be solved efficiently by column generation.

3 Column Generation

Column generation is the standard method to solve the LP relaxation of the set-partitioning model. The basic concept of by column generation is to dynamically generate routes which possibly consist of the optimal solution, instead of enumerating the whole set of feasible routes. Column generation consists of two subproblems: restricted master problem (RMP) and the pricing problem, which are iteratively solved until no negative reduced cost routes are founded.

In the remainder of this section, we first introduce the pricing problem and formulate it into an MIP model. Then we describe the label-setting algorithm in detail. The bounded bidirectional search strategy is inefficient in the SPDPLCTW due to the structural propriety. Last we present the tabu heuristic column generator to speed up the label-setting algorithm.

3.1 Pricing Problem

The pricing problem is a constrained elementary shortest path problem derived by applying Dantzig-Wolfe decomposition to the arc-flow model. Let π_i be the dual value of the constraint (17)

and (18) corresponding to node i, and μ be the dual value of the constraint (19). Let $x_{i,j}$ be a binary decision variable which is set to 1 if arc (i,j) is selected in the optimal route and 0 otherwise. Let a_i and w_i be non-negative decision variables representing the arrival time and load of the vehicle at node i respectively. The pricing problem is as follows:

$$min \sum_{(i,j)\in A} d_{i,j}x_{i,j} + \lambda \times \sum_{(i,j)\in A} w_i d_{i,j} x_{i,j} - \sum_{(i,j)\in A} \pi_i x_{i,j} - \mu \tag{21}$$

$$s.t. \sum_{(j,i)\in A} x_{j,i} = \sum_{(i,j)\in A} x_{i,j}, \forall i \in P \cup D \tag{22}$$

$$\sum_{(0,j)\in A} x_{0,j} = 1, \forall k \in K \tag{23}$$

$$\sum_{(i,n+1)\in A} x_{i,n+1} = 1, \forall\ k \in K \tag{24}$$

$$w_j \geq w_i + q_j + (x_{i,j} - 1)W, \forall (i,j) \in A \tag{25}$$

$$w_i \geq w_j - q_j + (x_{i,j} - 1)W, \forall (i,j) \in A \tag{26}$$

$$0 \leq w_0 \leq Q \tag{27}$$

$$w_{n+1} = 0 \tag{28}$$

$$a_j \geq a_i + d_{i,j} + (x_{i,j} - 1)W, \forall (i,j) \in A \tag{29}$$

$$e_i < a_i < l_i, \forall i \in P \cup D \tag{30}$$

$$a_{n+1} - a_0 \leq T \tag{31}$$

$$max(0, q_i) \leq w_i \leq min(Q, Q + q_i), \forall i \in P \cup D \tag{32}$$

$$x_{i,j} \in \{0,1\}, \forall (i,j) \in A \tag{33}$$

$$a_i \geq 0, \forall i \in V \tag{34}$$

The objective function (21) minimizes the reduced cost of the selected route, which consists of distance cost, loading cost and the collection of prices of dual variables. The rest constraints are derived from the corresponding constraints in the arc-flow model to ensure the feasibility of the generated route. The pricing problem is the bounded elementary shortest path problem which should satisfy the capacity, maximum duration, and time window constraints. However, the major difference distinguishing this problem from the pricing problem in the 1-1PDPLC is that this pricing problem is without precedence constraints between pickup and delivery. Because this pricing problem is different, we propose a new label-setting algorithm, which is detailed in the next section.

3.2 The label-setting Algorithm

Comparing SPDPLCTW with the previous 1-1PDPLC, we can clearly notice that the main difference is that the many-to-many commodity structure, which cause exponential increase of labels during the label extension. Another difference is the undetermined initial load form depot, which is adjusted step by step during extension and makes the label has a linear structure. Here I can develop a label-setting algorithm for the column generation in the SPDPLCTW.

Let $L_i = \{f_i, \tau_i, d_i, l_i, cl_i, N_i\}$ denote a label which represents a partial route *r* from node 0 to node i where:

f_i is the fix reduced cost of this partial route *r*;

τ_i is the arrival time of vehicle at node i;

d_i is the traveling distance of vehicle from depot 0 to node i;

l_i is the maximum initial load from depot that can be provided at node i and labels at depot are with initial load $l_0 = Q$;

cl_i is the accumulated loads from pickups at node i;

N_i is the set of nodes that have been visited in the partial route.

The extension rule is described as follow. Given a label $L_i = \{f_i, \tau_i, d_i, l_i, cl_i, N_i\}$ associating with node i, a new label $L_j = \{c_j, \tau_j, d_j, l_j, cl_j, N_j\}$ can be created by extending L_i through arc (i, j) if it satisfies following conditions:

$$l_i + cl_i + q_j \geq 0 \tag{35}$$

$$cl_i + q_j \leq Q \tag{36}$$

$$j \notin N_i \tag{37}$$

$$\tau_i + d_{i,j} \leq l_j \tag{38}$$

Condition (35) and (36) ensure that the partial path from *0* to i can satisfy the demand of node j and not exceed the vehicle capacity. Condition (37) forbids the extension to a node that has have been visited and condition (38) ensures the arriving time not violates the time window of node j. The elements in L_j can be determined according to the following extension rules:

$$\tau_j = \max\{\tau_i + d_{i,j}, e_j\} \tag{39}$$

$$d_j = d_i + d_{i,j} \tag{40}$$

$$N_j = N_i \cap \{ j \} \tag{41}$$

$$f_j = \begin{cases} f_i + (1+\lambda cl_i)d_{i,j} - \mu_j, if \ \ cl_i + q_j \geq 0 \\ f_i + (1+\lambda cl_i)d_{i,j} - \mu_j + (|\, cl_i + q_j \,|)d_j, if \ \ cl_i + q_j < 0 \end{cases} \tag{42}$$

$$cl_j = \begin{cases} cl_i + q_j, if \ \ cl_i + q_j \geq 0 \\ 0, if \ \ cl_i + q_j < 0 \end{cases} \tag{43}$$

$$l_j = \begin{cases} \min(l_i, Q - cl_j), if \ \ cl_i + q_j \geq 0 \\ l_i + cl_i + q_j, if \ \ cl_i + q_j < 0 \end{cases} \tag{44}$$

To accelerating the label-setting algorithm, we applied two methods to remove the abundant labels. The first method is to discard labels which cannot extend to form a feasible route. For a label L_i, if $\tau_i + d_{i,n+1} > T$, it can be removed. Second, the domination strategy is applied to eliminate labels. For a label L_i, we use $r(L_i)$ denoting the partial route defined by L_i, and let $R(L_i)$ denote the set of all feasible partial route starting from i that can be combined with $r(L_i)$ to form a feasible route. As we assume that for every feasible route there won't be any load carried back to the depot, the demand of successive partial route $r \in R(L_i)$ must be equal to or more than the accumulative load (cl_i) of partial route $r(L_i)$ at i, and less than the total supply load ($l_i + cl_i$), which is a restrict condition for $R(L_i)$. Assume two labels L_i^1 and L_i^2 associating with a same node i. Under this condition, if L_i^2 dominates L_i^1, the supply range of L_i^2 must overlap the range of L_i^1 to ensure $R(L_i^1) \subseteq R(L_i^2)$. The detail criteria for label $L_i^2 = \{ f_i^2, \tau_i^2, d_i^2, l_i^2, cl_i^2, N_i^2 \}$ to dominate $L_i^1 = \{ f_i^1, \tau_i^1, d_i^1, l_i^1, cl_i^1, N_i^2 \}$ are described as follow:

$$\tau_i^2 \leq \tau_i^1 \tag{45}$$

$$N_i^2 \subseteq N_i^1 \tag{46}$$

$$cl_i^2 \leq cl_i^1 \tag{47}$$

$$l_i^2 + cl_i^2 \geq l_i^1 + cl_i^1 \tag{48}$$

The constraints (45) – (48) ensure the set of successive partial route of label 1 is included by the set of label 2. The reduced cost of one partial route $r(L_i)$ is a linear segmental function $Z_i(q)$

of the unconsumed load q ($cl_i \le q \le l_i + cl_i$). As shown in figure 2, the supply range is from cl_i to $l_i + cl_i$, and the slope of the segment is d_i.

$$Z_i(q) = f_i + (q - cl_i)d_i, \ cl_i < q \le l_i + cl_i \tag{49}$$

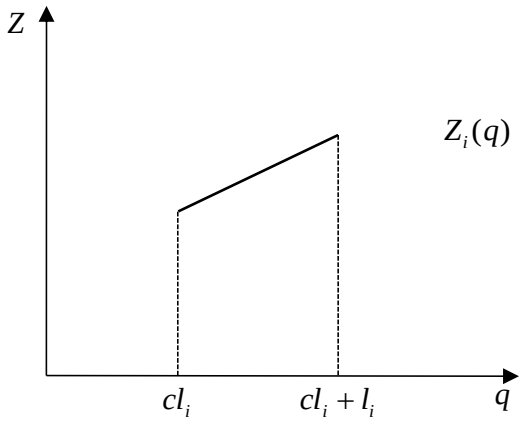


Figure 2 Reduced cost function of $Z_i(q)$

For the domination rule, to compare the reduced cost of two labels, we should compare the reduced costs of the two endpoints of the segment of L_i^1:

$$Z_i^2(cl_i^1) \le c_i^1 \tag{50}$$

$$Z_i^2(l_i^1 + cl_i^1) \le Z_i^1(l_i^1 + cl_i^1) \tag{51}$$

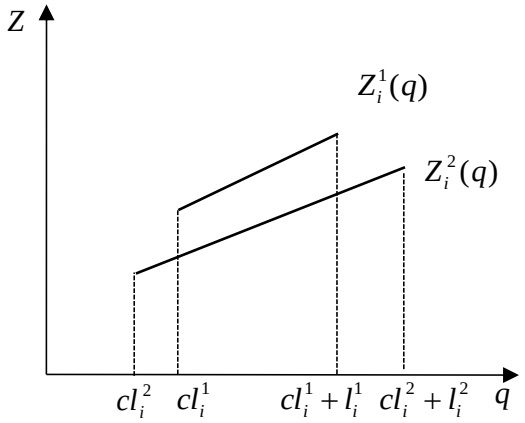


Figure 3 $Z_i^2(q)$ dominates $Z_i^1(q)$

3.3 Bounded Bidirectional Search

The bounded bidirectional search is a widely-used strategy to accelerate the label setting process. However, due to the unsymmetrical commodity matrix in SPDPLCTW, the bounded bidirectional search is inefficient. The proof is as follows.

The backward label-setting rule is similar with the forward rules. Let label $L_i^b = \{ f_i^b, \tau_i^b, cl_i^b, N_i^b \}$ denote a backward label of a partial route from i to n+1. The backward label holds the same configuration of forward label except that $cl_i^b \le 0$ and there is not the initial load l_i, as the vehicle goes back to terminal with empty load. Label L_j^b can extend from L_i^b through arc (i, j) under following conditions:

$$-Q \le cl_i^b + q_j \le 0 \tag{52}$$

$$j \notin N_i^b \tag{53}$$

$$\tau_i^b + d_{i,j} \ge e_j \tag{54}$$

The elements of L_i^b is configured as:

$$f_j^b = f_i^b + (1 - \lambda cl_i) d_{i,j} - \mu_j \tag{55}$$

$$\tau_j^b = \min\{\tau_i^b - d_{i,j}, l_j\} \tag{56}$$

$$cl_j^b = cl_i^b + q_j \tag{57}$$

$$N_j^b = N_i^b \cap j \tag{58}$$

For the domination rule, as the vehicles are restricted to go back to terminal with empty load, the backward partial route has a net demand that will be supplied in future extension. Therefore, the reduced cost of the backward partial route is a fix number. $L_i^{b,2} = \{ f_i^{b,2}, \tau_i^{b,2}, cl_i^{b,2}, N_i^{b,2} \}$ dominates $L_i^{b,1} = \{ f_i^{b,1}, \tau_i^{b,1}, cl_i^{b,1}, N_i^{b,1} \}$ in conditions:

$$\tau_i^{b,2} \ge \tau_i^{b,1} \tag{59}$$

$$N_i^{b,2} \subseteq N_i^{b,1} \tag{60}$$

$$cl_i^{b,2} = cl_i^{b,1} \tag{61}$$

$$f_i^{b,2} \le f_i^{b,1} \tag{62}$$

In Comparison to the forward domination rule, the backward domination rule is weak. This is mainly because that the vehicles have an initial load at the depot but are restricted to be empty when back to depot, which makes the SPDPLCTW contains an unsymmetrical commodity matrix. Due to this unsymmetrical propriety, the accelerating effect of bidirectional label setting algorithm has been greatly weakened in the SPDPLCTW. Therefore, the bounded bidirectional search is not applied in the SPDPLCTW.

3.4 Tabu Heuristic Column Generator

To accelerate the label-setting process, we can apply some heuristics to identify negative reduced cost columns [16,17,18,19], which takes much less computational time when compared with the exact label-setting algorithm. In this thesis, the tabu heuristic is developed as shown in Algorithm 4, to heuristically and rapidly identify negative reduced cost columns. At each iteration, we first use the tabu algorithm to solve the pricing problem. If it manages to obtain some columns with negative reduced cost, we add these columns into the matrix of the LMP and solve it. The tabu heuristic will repeat until no columns with negative reduced cost are found or the number of found columns exceeds a default number *MaxCol*. Then we invoke the exact label-setting algorithm to solve the pricing problem to optimality.

The tabu algorithm is a well-known heuristic algorithm based on local search mechanism, which is a basic concept to search a better solution in a small space around current solution. In our algorithm, we start with a set R_0 of routes with zero reduced cost in the optimal solution of the current LMP. Two type of local move are defined: (1) insert a node not in the route into the route (shown as Algorithm 2); (2) remove a node already in the route (shown as Algorithm 3). For each route in R_0, we repeat the local search *maxItr* times. In each cycle, we search the nearby space of current solution using insertion search and remove search to find a best nearby feasible solution, and change the current solution into the best nearby solution. Once we find a route with negative reduced cost, we add it into the *route pool*. There is a tabu mechanism to restrict repeat movement. After a movement is used in one cycle, this move cannot be used again in *expire* times, unless we can get a solution better than the recent solution. The whole tabu heuristic is shown in Algorithm 4.

Algorithm 2 Greedy insert (*r*, *i*)

```
r' ← null ;
for each pair of two consecutive vertices u and v in r do
   insert i between u and v;
   if resulting r is not feasible then
      Continue;
   else if r is not initialized then
      r' ← r;
   else if Z(r, 0) < Z(r', 0) then
      r' ← r;
   end if
   restore r to its state before insert;
end for

Return r'.
```

Algorithm 3 Greedy remove (r)

1: $r' \leftarrow null$;
2: **for** each $i \in r$ except 0 and $n + 1$ **do**
3: remove i in r;
4: **if** resulting r is not feasible **then**
5: **Continue**;
6: **else if** r is not initialized **then**
7: $r' \leftarrow r$;
8: **else if** $Z(r, 0) < Z(r', 0)$ **then**
9: $r' \leftarrow r$;
10: **end if**
11: restore r to its state before insert;
12: **end for**

Return r'.

Algorithm 4 Tabu Heuristic

Input: A set of routes R_0, *expire*, *maxItr*;
1: Define *route pool* as a set of route;
2: **for** route $r \in R_0$ **do**
3: Define $Tabu_list[i] = 0, \forall i \in P \cup D$;
5: $k \leftarrow 0$;
6: **while** $k < maxItr$ **do** $recent_cost = 100000$;
7: $vertex_queue \leftarrow V / \{\text{vertex in } r\}$,*option* $\leftarrow$ *null* and *option node* $\leftarrow$*null*;
8: **for** $i \in vertex_queue$ **do**
9: $r' \leftarrow Greedy\ Insert(r, i)$;
10: **if** $r' = null$ **then**
11: **Continue**;
12: **end if**
13: $cost \leftarrow Z(r', 0)$;
14: **if** $cost < recent_cost$ and $k - Tabu_list[i] > expire$ **then**
15: $recent_cost \leftarrow cost$, *option* $\leftarrow r'$ and *option node* $= i$;
16: **end if**
17: **if** $cost < Z(r, 0)$ and $cost < recent_cost$ **then**
18: $recent_cost \leftarrow cost$, *option* $\leftarrow r'$ and *option node* $= i$;
19: **end if**
20: **if** $cost < 0$ and $r' \notin$ *route pool* **then**
21: add r' into *route pool*;
22: **end if**
23: **end for**
24: **for** $i \in r/\{0,n+1\}$ **do**
25: $r' \leftarrow Greedy\ remove(i)$;
26: **if** $r' = null$ **then**
27: **Continue**;
28: **end if**
29: $cost \leftarrow Z(r', 0)$;
30: **if** $cost < recent_cost$ and $k - Tabu_list[i] > expire$ **then**
31: $recent_cost \leftarrow cost$, *option* $\leftarrow r'$ and *option node* $= i$;
32: **end if**
33: **if** $cost < Z(r, 0)$ and $cost < recent_cost$ **then**
34: $recent_cost \leftarrow cost$, *option* $\leftarrow r'$ and *option node* $= i$;
35: **end if**
36: **if** $cost < 0$ and $r' \notin$ *route pool* **then**
37: add r' into *route pool*;
38: **end if**
39: **end for**
40: $r \leftarrow$ *option*, $Tabu_list[$*option node*$] = k$ and $k \leftarrow k + 1$
41: **end while**
42: **end for**
Return *route pool*;

4 Branch-and-Price-and-Cut Algorithm

Branch-and-price-and-cut is one of the leading solution procedure for many large-scale integer programming models [20,21], which can be seen as a combination of the branch-and-price and branch-and-cut algorithm. In the branch-and-price-and-cut algorithm, at each node on the branch-and-bound tree, a lower bound is firstly obtained by solving the LMP based on the column generation procedure. For the node that cannot be pruned, try to identify if the current linear solution violates some valid inequalities. If such violated inequalities are found, add them into the model and invoke the column generation procedure again to improve the lower bound. The above procedure is repeated until the node is pruned or no violated inequalities can be found. In this section, we first describe several valid inequalities brought from other related problems. Next, the initial solution and upper bound, search and branching strategies are presented.

4.1 Valid Inequalities

As SPDPLCTW is a totally new problem, I tested several famous inequalities brought from some close problems: *clique cluster inequality* and *homogeneous multistar inequality* from the 1-PDTSP; *Rounded capacity inequality* from the PDPTW. These inequalities are defined on the master problem variables θ_r. The detail is presented as follows:

Inequalities from the 1-PDTSP: Clique cluster inequality and homogeneous multistar inequality were introduced by Hernández-Pérez and Salazar-González [35] to solve the 1-PDTSP. The clique inequality is generated from the stable set problem, and relies on the vehicle capacity constraint. Let $v \in D$ be a delivery node and a collection of delivery customer subsets $W_1,...,W_m$ satisfy following conditions:

$$W_i \cap W_j = v, 1 \le i < j \le m \tag{63}$$

$$r(W_i) = 1, i \in 1,...,m \tag{64}$$

$$r(W_i \cap W_j) > 1, 1 \le i < j \le m \tag{65}$$

Let $E(S) := \{(i, j) \in E, i, j \in S\}$ indicate arc in *S* and $x(E(s)) = \sum_{k \in K} \sum_{(i,j) \in E(S)} x_{i,j,k}$ indicate the sum of active edges in set *S*. $r(S)$ is the minimal number of vehicle required to satisfy the nodes in set S. Because it's only necessary to visit deliveries in SPDPLCTW, $r(S)$ is set as:

$$r(S) = \left\lceil - \sum_{i \in S \cap D} q_i / Q \right\rceil \tag{66}$$

The clique cluster inequality is:

$$\sum_{i=1}^{m} x(E(W_i)) \le \sum_{i=1}^{m} | W_i | - 2m + 1 \tag{67}$$

Hernández-Pérez and Salazar-González [35] also developed the homogeneous multistar inequality that transformed from the CVRP multistar inequalities. Let $N \in V$ be the nucleus set and $S \in V$ be the satellite set. Due to the propriety of SPDPLCTW, these two sets only consist of deliveries nodes. Let $E(N) := \{(i, j) \in E, i, j \in N\}$ indicate the set of edges within *N* and $E(N:S) := \{(i, j) \in E, i \in S, j \in N \ or \ i \in N, j \in S\}$ indicate the set of edges connecting *N* and *S*. Then the homogeneous multistar inequality is:

$$\lambda x(E(N)) + x(E(N:S)) \leq \mu \tag{68}$$

The constraint (68) is valid for appropriate value of λ and μ. Let $x(E(N:S)) = \alpha$ and $UB(\alpha)$ the upper bound of $x(E(N))$, which constructs a polygon in $(\alpha, UB(\alpha))$ space. The value of λ and μ can be found by a calculating the convex hull of the polygon in $(\alpha, UB(\alpha))$ space. However, the separation algorithms introduced by Hernández-Pérez and Salazar-González [35] cannot find any violated clique cluster inequality and homogeneous multistar inequalities in almost all instances of SPDPLCTW. One explanation may be the strong lower bound obtained by the set-partitioning model, as the two inequalities were applied to the arc-flow model of the 1-TSPPD. Another reasonable explanation is that because the all these inequalities are based on the vehicle capacity constraint. In the SPDPLCTW, all these inequalities can only be applied to the delivery customers, which greatly weakens the feasibility of these two inequalities.

Rounded Capacity Inequality from the PDPTW: Rounded capacity inequality is a famous inequality applied in the VRP researches and it can be expressed as:

$$\sum_{r \in \Omega} \sum_{(i,j) \in E^{-1}(S)} \beta_{i,j,r} \theta_r \geq r(S), \forall S \in \Gamma \tag{69}$$

Let Γ be the set of all customer subset $S \in V_c$. $\beta_{i,j,r}$ is a binary parameter, where $\beta_{i,j,r} = 1$ if edge (i, j) is used in r and $\beta_{i,j,r} = 0$ if not. Let $E^{-}(S) = \{(i, j) \in E, i \in S, j \notin S\}$ indicate the set of all the edge leaving node set *S*. Because the rounded capacity inequality is also based on the vehicle capacity constraint, the rounded capacity inequality can only apply to delivery nodes in SPDPLCTW. Let $r(S)$ be the minimal number of vehicle required to serve the set *S* which is set as (66).

To identify the violation of rounded capacity inequalities, a greedy insertion heuristic proposed by Ropke and Cordeau [34] is used as the separation algorithm, the violated inequalities can be found in most instances. Therefore, the rounded capacity inequality is adopted in our algorithm. Let $\gamma = \{\gamma_{s_1}, \gamma_{s_2}, ..., \gamma_{s_{|\Gamma|}}\}$ indicate the value of dual variable associated with inequality (69). The reduced cost of a route *r* can be reformed as：

$$\sum_{(i,j)\in A} d_{i,j}x_{i,j} + \lambda \times \sum_{(i,j)\in A} w_i d_{i,j}x_{i,j} - \sum_{(i,j)\in E} \pi_i x_{i,j} - \mu - \sum_{S\in\Gamma} \sum_{(i,j)\in S\cap r} \gamma_S \qquad (70)$$

The extension rule of fix cost should be modified as:

$$\bar{f}_j = \begin{cases} \bar{f}_i + (1+\lambda cl_i)d_{i,j} - \mu_j - \sum_{S\in\Gamma:(i,j)\in E^-(S)} \gamma_S, if\ \ cl_i + q_j \geq 0 \\ \bar{f}_i + (1+\lambda cl_i)d_{i,j} - \mu_j + (|\, cl_i + q_j\,|)\tau_j - \sum_{S\in\Gamma:(i,j)\in E^-(S)} \gamma_S, if\ \ cl_i + q_j < 0 \end{cases} \qquad (71)$$

To limit the overall computational time, the separation of rounded capacity inequalities is only applied at a node whose depth in the branch and bound tree is less than 2 (Assuming the root node is at depth 0). In the separation procedure, the rounded capacity inequalities are identified when the constraint (69) are violated by more than a predetermined threshold value (0.05 in our tests). Then the violated constraint (69) is added into the LMP to reoptimize the lower bound.

4.2 Initial Solution

The initial feasible solution is generated by a greedy insertion heuristic algorithm, which iteratively construct routes to satisfy all the deliveries. Each time, the insertion procedure starts with an empty route which departs from 0 and directly ends to n+1. Firstly, an unserved delivery is randomly selected to insert at a feasible place of the incumbent route in which the increasing cost is smallest. Then all unused pickups are inserted at every feasible place of the incumbent route to minimize the route cost. The insertion process repeats until no more nodes can be inserted into incumbent route. After this, the heuristic adds the incumbent route into the solution and repeats the process with a new empty route. The whole algorithm ends when all deliveries are served.

4.3 Search Strategy

The branch-and-bound tree is explored according to a best-first policy. The unexamined nodes are sorted in increasing order according to the lower bounds of their fathers and stored in a priority queue. Every time the first node in the priority queue is popped out and solved by the column generation approach.

4.4 Branch Strategies

Following the branch strategy for branch-and-price algorithm introduced by Desaulniers et al. [22], where first branch on the total number of vehicles if it is fractional, and then branch on arcs which have fractional values. Let $\bar{\theta}_r (r \in \Omega)$ denote the optimal LP solution and $\rho = \sum_{r\in\Omega} \bar{\theta}_r$ be the number of vehicles used in this solution. If ρ is fractional, we branch the current node into

two child nodes: one has constraint $\sum_{r\in\Omega}\theta_r \le \lceil \rho \rceil$ and the other has constraint $\sum_{r\in\Omega}\theta_r \ge \lfloor \rho \rfloor$. These two branching constraints can be enforced easily by changing the lower bound or upper bound of constraint (19).

Let $\beta_{i,j,r}$ be a binary number which is equal to 1 if route r covers arc (i, j) and 0 otherwise. Let $\bar{x}_{i,j} = \sum_{r\in\Omega}\beta_{i,j,r}\bar{\theta}_r$ be the value of arc (i, j). Select the arc (i, j) which has $\bar{x}_{i,j}$ closest to 0.5 to branch on. The current node is branched into two child nodes: one with constraint $\sum_{r\in\Omega}\beta_{i,j,r}\theta_r = 0$ and the other with constraint $\sum_{r\in\Omega}\beta_{i,j,r}\theta_r = 1$. In the case where $\sum_{r\in\Omega}\beta_{i,j,r}\theta_r = 0$, arc (i, j) cannot be used in any route and this can be enforced by forbidding extension along arc (i, j). Similarly, in the case where $\sum_{r\in\Omega}\beta_{i,j,r}\theta_r = 1$, arc (i, j) should be used once and consequently arcs (i,k) $(k \ne j)$ and (h, j) $(h \ne i)$ are forbidden. Therefore, this branching constraint can be enforced by forbidding extension along arcs (i,k) $(k \ne j)$ and (h, j) $(h \ne i)$. Note that after branching on arcs, the resultant graph is no longer a complete graph.

We also tested to branch on the visit time of pickups due to the selective pickups of the SPDPLCTW. Let $\bar{o}_j = \sum_{r\in\Omega}\sum_{i\in V}\beta_{i,j,r}\bar{\theta}_r$ be the visit time of pickup *j* in the optimal solution. Select the pickup i which has $\bar{o}_i$ closest to 0.5 to branch on. The current node is branched into two child nodes: one with constraint $\sum_{r\in\Omega}\alpha_i\theta_r = 0$, where the extension to pick-up i is forbidden in label-setting algorithm and the other with constraint $\sum_{r\in\Omega}\alpha_i\theta_r = 1$, which replaces the corresponding constraint (18). However, the introduction of branching on pickup visit time doesn't reduce the numbers of branch and bound nodes and the computational time. One explanation is its high closed to the branching on arcs.

5 Computational Experiments

5.1 Instance Generation

The experimental instances are generated by the classical data generator presented in Ropke and Cordeau [34] for the PDPTW, and the one-to-one correspondence between pickup and delivery is released in the SPDPLCTW. The coordinates of customers are randomly selected within a [0, 50] × [0, 50] square according to a uniform distribution and the depot is located in (25, 25). The number of available vehicles |K| is set to be $N / 2$, where *N* is the number of customers. The time window of each customer is constructed by randomly selecting a_i in $[0, T - d_{i,n+1}]$ and setting $b_i = a_i + W$. Same as the Ropke's generator, the time window interval W is set by 60, the vehicle

capacity Q is 20, the vehicle maximum duration T is set by 600. Following Luo et al. [72], the coefficient λ is set by 1.0 for all of the test instances. The only difference from the Ropke's data generator is the generation of the node demand. The demands q_i are randomly selected in the interval [5, Q] and are transferred to be negative when i is a delivery.

5.2 Experiment Setup

All algorithms were coded in Java and all experiments were conducted on a Dell server with an Intel Xeon E5520 2.26 GHz CPU, 8 GB RAM and Linux operating system. The linear programming model RLMP was solved by simplex algorithm implemented by ILOG CPLEX 12.0. Computational times reported are in CPU seconds on this server. We imposed a time limit of 36,00 seconds on each run of the branch-and-price-and-cut algorithm. For the tabu heuristic accelerating algorithm, the parameters used were fixed as: *expire* = 20, *maxItr* = 60 and *maxCol* = 5000. According to some preliminary experiments, we observed that the value of *maxCol* has the main impact on the performance of the tabu heuristic.

According to the number of pickups and deliveries, the instances in our experiment can be divided into nine classes, which have 10, 20, 30 pickups and 20, 40, 60 deliveries, respectively. Each class has five instances and, hence, there are 45 instances in total. The class of instances with 10 pickups and 10 deliveries are designed for the branch-and-cut algorithm realized by the famous optimization software LOG CPLEX 12.0 to compare and illustrate the efficiency of proposed branch-and-price-and-cut algorithm. Because the demands of customers are dependent on the vehicle capacity Q , we do not conduct computational experiments on the influence of Q by assigning different values for it. We first test the impact of the tabu heuristic on the performance of label-setting algorithm. Then the branch-and-price-and-cut algorithm is applied to solve all instances. Last, we illustrate the impact of the selective pickups on the cost of the optimal solutions. Note that when there are no selective pickups, the SPDPLCTW is reduced into the vehicle routing problem with loading cost and time window (VRPLCTW). To provide a better understanding of the cost saving effect of selective pickups, we use the solutions of the VRPLCTW as the benchmark.

5.2 Impact of Tabu Heuristic

We first test the impact of the tabu heuristic in the label setting algorithm. The computational results of all instances show in the table 1. The column *P#* and *D#* are separately the numbers of pickup customers and delivery customers; column ν is the value of the supply parameter. The columns *Time(s)* give the computational time in seconds to obtain the optimal LP solutions and the columns *TabuTime(s)* give the computational time in seconds of tabu heuristic. Columns *Label* give the number of labels generated by the label-setting algorithm in the last iteration. As showed in the table 1, the solve time is increasing along the number of pickups, which demonstrates the complexity of problem increases with the number of pickups. And the branch- and-price with tabu

heuristic only outperforms the branch-and-price in the instances with 20 and 30 pickups, which means that the accelerating effect of tabu heuristic is only significant for the large instances.

Table 1: Impact of the tabu heuristic column generator

P#	D#	Inst	Branch and Price		Branch and Price with Tabu		
			Label#	Time(s)	Label#	TabuTime(s)	Time(s)
10	20	D06	159396	0.38	41752	0.31	0.48
		D07	219146	0.47	3053	0.41	0.43
		D08	114224	0.34	22798	0.31	0.40
		D09	125805	0.28	12245	0.38	0.46
		D10	116177	0.31	23325	0.31	0.42
10	40	D11	869095	1.00	120135	1.21	1.52
		D12	289141	0.46	73906	0.69	0.91
		D13	378973	0.81	44567	0.78	0.98
		D14	574045	0.82	193704	0.56	0.91
		D15	574039	0.66	77534	0.97	1.23
10	60	D16	1452067	1.07	73963	1.50	1.77
		D17	807269	0.88	115588	1.74	2.11
		D18	1865135	1.40	426508	1.74	2.20
		D19	2166821	1.27	498264	2.05	2.63
		D20	907049	1.14	67619	1.58	1.88
	Average		707892.1	0.75	119664.1	0.97	1.22
20	20	D21	563720	0.67	95150	0.57	0.90
		D22	1045572	1.10	219128	0.55	1.03
		D23	1516510	2.22	134205	0.89	1.40
		D24	374092	0.62	54140	0.50	0.68
		D25	1475419	2.10	839954	0.39	1.72
20	40	D26	4635939	4.37	794813	2.09	3.64
		D27	6264467	3.42	774246	3.15	3.76
		D28	1218235	1.60	204450	2.37	2.87
		D29	4605999	2.62	2214245	2.83	4.28
		D30	9552798	10.93	1266132	1.51	2.48
20	60	D31	7210096	4.40	2517387	4.66	6.54
		D32	8293460	5.69	1721588	3.78	5.47
		D33	11354525	7.00	2004069	6.23	8.15
		D34	19162960	25.38	3722326	5.28	11.52
		D35	7056162	5.63	373932	4.48	5.24
	Average		5621996.9	5.18	1129051.0	2.62	3.98
30	20	D36	2855961	4.69	524546	0.53	1.20
		D37	4747167	7.01	553488	2.85	3.26
		D38	3209678	5.49	529931	1.24	2.11
		D39	1753053	3.41	374676	0.72	1.48
		D40	1370853	1.14	553695	1.38	2.08
30	40	D41	15389131	15.74	4901756	5.84	13.85
		D42	27680373	21.86	8397243	8.79	17.77
		D43	7035659	4.70	2454983	2.66	4.55
		D44	11188287	8.14	5933652	5.29	11.33
		D45	18305579	22.94	4047792	3.55	11.55
30	60	D46	54044840	110.55	16331225	10.43	78.18
		D47	28194750	17.84	8481834	17.71	24.63
		D48	23155800	21.93	4328143	10.62	16.33
		D49	18212328	24.78	1814849	10.83	12.97
		D50	33582494	41.48	7597725	13.11	24.53
	Average		**16715063.5**	**20.78**	**4455035.9**	**6.37**	**15.05**

5.3 Optimal Solution Results

We first compare the results between the branch-and-price-and-cut algorithm and branch and cut algorithm realized by the LOG CPLEX 12.0. The result is showed in the table 2. As most of the instances with more pickups and deliveries are unsolved by the CPLEX, we only present these small instances with 10 pickups and 10 deliveries. The column *P#* and *D#* separately give the number of pickups and deliveries of instances. The column *LB* gives the lower bound obtained at the root node and the column *Node#* gives the number of node searched in the branch and bound tree. The column *Time(s)* gives the total computational time. The results suggest that the branch-and-price-and-cut algorithm substantially outperforms the branch-and-cut algorithm by a large margin, which demonstrates the superiority of the branch-price-and-cut algorithm in solving the SPDPLCTW.

Table 2: Comparison between branch-and-price-and-cut and branch-and-cut

P #	D #	Inst	Branch-and-cut			Branch-and-price-and-cut		
			Node#	LB	Time(s)	Node#	LB	Time(s)
10	10	D01	105764	1101.5	157.27	1	1435.2	0.40
		D02	711802	894.7	992.17	1	1263.9	0.45
		D03	505387	1342.2	579.25	1	1773.1	0.31
		D04	762793	794.9	877.9	1	1569.4	0.23
		D05	359332	1069.4	411.66	5	1513.5	0.49
	Average		489015.6	1040.5	603.65	1.8	1511.0	0.376

Table 3 shows the results of all other instances and the corresponding columns of unsolved instances within the time limit in table 3 are fulfilled by Hyphen -. Based on previous test, the tabu heuristic is only applied for instances with 20 and 30 pickups. The column *UB* gives the value of the initial solution of this instance. The column *LB* and *IP* give the value of lower bound of the root node and the value of the optimal solution, separately. The optimal gaps, which are defined as $100(IP - LB)/IP$, are presented in the column *Gap*. The average value of *Gap* is 0.11%, which demonstrates the high efficiency of the lower bound obtained by the set-partitioning model. As the proposed branch-and-price-and-cut algorithm is obviously suitable for the VRPLCTW, we can take the VRPLCTW as the benchmark. The column *VRP* and *VRP time* give the value of the optimal solution of the instance without selective pickups (VRPLCTW) and the corresponding solving time realized by the proposed algorithm. All instances without selective pickups can be solved quickly, which suggests the SPDPLCTW is more complicated than the VRPLCTW. The column *Save* is defined as $100(VRP\text{-}IP)/VRP$, which indicates the effect of saving cost by introducing selective pickups. Columns *Cap* gives the number of rounded capacity inequality and columns *Use* gives the number of the used pickups in the optimal solution. Column *Tabu time* and *LS time* give the time spent by tabu heuristic and label-setting algorithm, respectively. And the column *Time* gives the total computational time. All instances with 10 and 20 pickups can be solved to optimality within the time limit. Otherwise, some instance with 30 pickups and 40 or 60 deliveries are unsolvable within time limit. The result suggests the complexity of the SPDPLCTW increases with the number of pickups and delivery customers. In terms of computational time, the time spent in the tabu heuristic column generator and the label-setting algorithm consists of a large proportion, which suggests that the pricing problem is still the bottle neck of the algorithm.

Table 3: Optimal solutions obtained by branch-and-price-and-cut

P#	D#	Inst	LB	Gap	IP	UB	VRP	Save	Node	Cap	Use	VRP time	Tabu time	LS time	Time
10	20	D06	3497.5	0.00%	3497.5	5515.2	3963.4	11.76%	1	0	7	0.39	-	0.41	0.60
		D07	2806.0	0.00%	2806.0	4636.9	3273.6	14.28%	1	0	8	0.34	-	0.49	0.68
		D08	2419.1	0.30%	2426.3	4940.3	2980.8	18.60%	47	2	6	0.23	-	2.04	3.55
		D09	2856.8	0.12%	2860.3	5584.9	3388.2	15.58%	3	0	8	0.17	-	0.57	0.76
		D10	2664.0	0.13%	2667.5	5017.5	3235.5	17.56%	7	0	7	0.29	-	1.22	1.62
10	40	D11	5615.4	0.00%	5615.4	9038.7	6720.5	16.44%	1	0	9	2.16	-	1.76	1.94
		D12	5074.9	0.00%	5074.9	7358.3	5835.5	13.03%	1	2	9	1.54	-	1.68	1.88
		D13	5698.5	0.00%	5698.5	9162.2	6476.8	12.02%	1	0	8	1.58	-	1.21	1.39
		D14	5720.8	0.00%	5720.8	7147.8	6483.0	11.76%	1	0	7	5.76	-	1.54	1.71
		D15	5560.6	0.05%	5563.6	7976.8	6443.0	13.65%	20	3	10	1.5	-	13.89	14.56
10	60	D16	8518.2	0.06%	8523.5	12312.3	9521.2	10.48%	29	8	9	16.67	-	79.87	81.00
		D17	9543.9	0.00%	9543.9	12905.7	10318.6	7.51%	1	4	8	111.59	-	19.29	19.55
		D18	8188.1	0.32%	8214.4	11418.5	9180.6	10.52%	311	4	10	111.73	-	202.71	211.51
		D19	7387.3	0.04%	7390.2	10956.5	8382.8	11.84%	1013	12	10	143.14	-	694.61	727.65
		D20	8110.0	0.05%	8113.9	11324.2	9039.0	10.23%	11	1	9	55.21	-	65.19	65.78
20	20	D21	2714.3	0.78%	2735.8	5312.7	3842.2	28.80%	9	0	9	0.50	1.85	0.64	3.44
		D22	2578.2	0.00%	2578.2	5033.8	3865.2	33.30%	1	0	14	0.91	0.59	0.50	1.55
		D23	2650.7	0.00%	2650.7	6099.1	4000.0	33.73%	1	0	12	0.29	0.94	0.53	2.05
		D24	2469.0	0.00%	2469.1	4820.7	3387.9	27.12%	3	0	12	3.97	0.92	0.42	1.86
		D25	2599.2	0.39%	2609.5	5997.4	3511.5	25.69%	7	0	8	0.27	1.31	2.84	4.94
20	40	D26	4755.5	0.26%	4768.1	7511.8	6296.9	24.28%	51	0	17	6.58	42.18	48.15	108.57
		D27	5054.8	0.34%	5071.9	9553.2	6876.9	26.25%	31	0	16	12.04	27.65	26.67	71.75
		D28	4779.6	0.17%	4787.6	8203.1	6282.2	23.79%	27	3	16	1.88	20.99	6.08	43.52
		D29	4532.9	0.00%	4532.9	7789.2	6123.1	25.97%	1	0	19	1.98	2.79	1.44	6.68
		D30	5315.0	0.06%	5318.3	8804.7	6580.9	19.19%	59	4	17	14.08	44.26	37.89	106.44
20	60	D31	7081.2	0.20%	7095.7	10999.4	9002.7	21.18%	209	6	20	16.39	388.71	328.65	914.89
		D32	8036.3	0.05%	8040.5	10922.3	9835.5	18.25%	25	0	18	111.36	51.98	36.05	216.02
		D33	7840.6	0.08%	7846.6	11760.7	9659.3	18.77%	53	7	19	14.48	107.89	110.84	370.45
		D34	7728.4	0.16%	7740.8	11757.6	9541.1	18.87%	177	8	18	23.66	370.73	1529.44	2104.18

Table 3 – *Continued from previous page*

P#	D#	Inst	LB	Gap	IP	UB	VRP	Save	Node	Cap	Use#	VRP time	Tabu time	LS time	Time
		D35	7513.6	0.04%	7516.7	11213.8	9010.1	16.57%	169	17	16	21.39	300.49	56.90	558.72
30	20	D36	1863.2	0.00%	1863.2	3132.7	3020.4	38.31%	1	0	11	0.3	0.56	0.70	1.67
		D37	2095.8	0.00%	2095.8	5514.9	3451.2	39.27%	1	0	16	0.44	3.01	0.49	5.39
		D38	2420.9	0.06%	2422.4	4928.5	3534.5	31.46%	11	0	15	0.29	3.72	5.83	10.89
		D39	2268.5	0.00%	2268.5	5078.9	3366.4	32.61%	1	0	13	0.26	0.75	0.85	2.06
		D40	2035.2	0.05%	2036.3	5109.2	3346.9	39.16%	5	0	14	0.26	2.50	1.41	5.09
30	40	D41	4388.7	0.00%	4388.7	7052.0	6044.1	27.39%	1	0	23	2.35	5.78	7.59	17.91
		D42	3891.5	0.67%	3917.9	9339.2	6325.1	38.06%	53	0	25	3.63	70.77	286.66	392.52
		D43	3978.1	0.00%	3978.1	7140.7	6374.1	37.59%	1	0	22	3.62	2.74	1.84	6.98
		D44	3862.5	-	-	9078.1	-	-	-	-	-	2.35	-	-	-
		D45	4814.7	0.14%	4821.3	8540.1	6557.4	26.48%	7	0	20	1.37	11.03	59.95	81.55
30	60	D46	6328.5	-	-	10742.3	-	-	-	-	-	154.33	-	-	-
		D47	6861.1	0.02%	6862.5	12651.4	9724.1	29.43%	9	0	29	155.22	44.46	60.33	204.51
		D48	7686.5	-	-	12650.9	-	-	-	-	-	38.24	-	-	-
		D49	6773.7	0.02%	6775.3	13305.8	9084.6	25.42%	15	1	26	18.83	39.76	29.10	175.64
		D50	6486.5	-	-	13753.6	-	-	-	-	-	127.19	-	-	-

As showed in the table 4, the complexity of SPDPLCTW increases along the number of pickup and delivery customers. Column *save* gives the average *Save* ratio, which reflects the saving cost effect of the selective pickups. The average saving ratios is 13.0% for instances with 10 pickups and is 32.0% for instances with 30 pickups, which suggests the transportation cost decreases with an increase in selective pickups. Let *S* give the average number of used pickups in the optimal solution. The average number of used pickups increases with the number of pickup customers and delivery customers. What's more, in all combinations, even for the instances with 10 pickups and 60 deliveries, the average number of used pickups is always less than the number of the selective pickups. This result suggests that the utilization of the selective pickups partially depends on their locations, and some pickups would never be visited since their remote locations.

Table 4: Impact of the selective pickups

P#	D# = 20				D# = 40				D# = 60			
	N	S	Time	Save	N	S	Time	Save	N	S	Time	Save
10	5	7.2	1.44	15.6%	5	8.6	4.3	13.4%	5	9.2	221.1	10.1%
20	5	11.0	2.77	29.7%	5	17.0	67.39	23.9%	5	18.2	832.85	18.7%
30	5	13.8	5.02	36.2%	4	22.5	124.74	32.4%	2	27.5	190.08	27.4%

6 Conclusion

The selective pickup and delivery problem with loading cost and time window (SPDPLCTW) is motivated by the *station model* applied by B2C businesses, such as *Alibaba* and *Ririshun*. The loading cost is considered in the context of the many-to-many PDP for the first time in literature. We first formulate the compact arc-flow model and the set-partitioning model of the SPDPLCTW. To solve the problem exactly, a branch-and-price-and-cut algorithm is devised based on the set-partitioning model. In the branch-and-price-and-cut algorithm, an ad hoc label-setting algorithm with linear structure is proposed to solve the pricing problem. Due to the unsymmetrical commodity structure, the bounded bidirectional search is inefficient to the SPDPLCTW. A tabu search heuristic is applied to improve the efficiency of the label-setting algorithm. Some related inequalities in the literature are tested in the set-partitioning model, but only the rounded capacity inequality is significant. The proposed algorithms were tested on a set of instances generated by a data generator derived from the generator in Ropke and Cordeau [15]. Experiment results show that the tabu heuristic substantially speeds up the label-setting algorithm in the large instances, and the cost of optimal solution significantly decreases with an increase in selective pickups. The proposed algorithm can efficiently solve instances with 30 pickups and 60 deliveries within time limit.

References

1. Anily S, Hassin R. The Swapping Problem[J]. Networks. 1992,22(4):419-33.
2. Wang F, Lim A, Xu Z. The one-commodity pickup and delivery travelling salesman problem on a path or a tree[J]. Networks. 2006,48(1):24-35.
3. Anily S, Gendreau M, Laporte G. The preemptive swapping problem on a tree[J]. Networks. 2011,58(2):83–94.
4. Anily S, Gendreau M, Laporte G. The swapping problem on a line[J]. Siam Journal on Computing. 1999 Sep 22,29(1):327-35.
5. Hernándezpérez H, Salazargonzález JJ. Heuristics for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem[J]. Transportation Science. 2004,38(2):245-55.
6. Hernández-Pérez H, Rodríguez-Martín I, Salazar-González JJ. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem[J]. Comput Oper Res. 2009,36(5):1639-45.
7. Hernández-Pérez H, Salazar-González JJ. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery[J]. Discrete Applied Mathematics. 2004,145(1):126-39.
8. Hernández-Pérez H, Salazar-González JJ. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms[J]. Networks. 2007,50(4):258-72.
9. Chalasani P, Motwani R. Approximating Capacitated Routing And Delivery Problems[J]. Siam Journal on Computing. 1998,28(6):21349.
10. Anily S, Bramel J. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries[J]. Naval Research Logistics. 1999,46(6):654-70.
11. Liao XL, Ting CK. An evolutionary approach for the selective pickup and delivery problem. Evolutionary Computation; 2010.
12. Liao XL, Ting CK. Evolutionary algorithms using adaptive mutation for the selective pickup and delivery problem. Evolutionary Computation; 2012.
13. Huang Y-H, Ting C-K. Genetic algorithm with path relinking for the multi-vehicle selective pickup and delivery problem. Evolutionary Computation (CEC), 2011 IEEE Congress on; 2011: IEEE.
14. Liao XL, Ting CK. Solving the biobjective selective pickup and delivery problem with memetic algorithm. Computational Intelligence in Production and Logistics Systems; 2013.
15. Ropke S, Cordeau JF. Branch and cut and price for the pickup and delivery problem with time windows[J]. Transportation Science. 2009,43(43):267-86.
16. Luo Z, Qin H, Zhu W, et al. Branch-and-price-and-cut for the Split-collection Vehicle Routing Problem with Time Windows and Linear Weight-related Cost[J]. Computer Science. 2014.
17. Athanasopoulos T, Minis I. Efficient techniques for the multi-period vehicle routing problem with time windows within a branch and price framework[J]. Annals of Operations Research. 2013,206(1):1-22.
18. Archetti C, Bouchard M, Desaulniers G. Enhanced branch and price and cut for vehicle routing with split deliveries and time windows[J]. Transportation Science. 2011,45(3):285-98.
19. Ribeiro GM, Desaulniers G, Desrosiers J. A branch-price-and-cut algorithm for the workover rig routing problem[J]. Transportation Science. 2015,39(12):3305-15.

20. Luo ZX, Qin H, Lim A. Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints[J]. European Journal of Operational Research. 2014 Apr 1,234(1):49-60.
21. Duhamel C, Potvin JY, Rousseau JM. A tabu search heuristic for the vehicle routing problem with backhauls and time windows[J]. Transportation Science. 1997 Feb,31(1):49-59.
22. Desaulniers G, Desrosiers J, Loachim I, et al. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems: Springer US; 1998. 57-93 p.

MISTA 2019

An Improved Opposition-based Differential Evolution with Gaussian Perturbation

Shuai Feng • Qing Xiong • Xinchao Zhao

Abstract Opposition-based Learning (OBL) is a new concept in machine learning, and it has been proven to be an effective method to some population-based algorithms in solving various optimization problems. Opposition-based differential evolution (ODE) has become one of successful applications of OBL, relying on its faster and more robust convergence than classical differential evolution (DE). ODE is a strict point-to-point algorithm, which may lead to neglect of points that are closer to the global optimal solution in the neighborhood of the opposite point. It appears that ODE provides a relatively narrow search area for the population candidate solutions and is not enough to increase population diversity. Thus, a new approach, namely GODE, is proposed to implement Gaussian perturbation on the opposite point in attempt to expand the search neighborhood of the opposite point and increase the possibility of finding a better solution. Experimental comparisons are conducted on CEC 2014 benchmark suite with 9 functions, and the results indicate that GODE is more competitive than DE and ODE.

1 Introduction

Differential Evolution (DE) was proposed by Price and Storn [1], which is a stochastic population-based optimization algorithm. Owing to its effectiveness, simplicity, robustness and a few control parameter, it has been successfully used in many applications, such as image processing [2], power systems [3], and filter design [4]. Similar with other evolutionary algorithms (EAs) [5, 6], it use mutation, crossover and selection operations at each generation to move its swarm toward even promising areas. Nevertheless, the classical DE tends to suffer

Shuai Feng
School of Science, Beijing University of Posts and Telecommunications Beijing 100876 China
E-mail: fengshuai11@bupt.edu.cn

Qing Xiong
School of Science, Beijing University of Posts and Telecommunications Beijing 100876 China
E-mail: xiongqing15@163.com

Xinchao Zhao
School of Science, Beijing University of Posts and Telecommunications Beijing 100876 China
E-mail: xcbupt@126.com

from premature convergence when solving complex problems. Consequently, many DE variants have been developed to improve the optimizing performance. A recent review on DE is provided in [7].

Opposition-based learning (OBL) was originated by Tizhoosh [8] in order to increase swarm diversity. The main idea behind OBL is considering the current search point and its opposite point simultaneously. In this way, it is possible to get a better approximation to the global optimum. It has been successfully applied to Reinforcement Learning (RL) [9], Artificial Neural Networks (ANN) [10], Fuzzy Systems [11] and other intelligent optimization algorithms. Among them, ODE is an outstanding representative of successful employing OBL for optimization problems, which was introduced in [12]. And after that, Rahnamayan *et al.* [13] presented the centroid opposition-based different evolution (CODE), which was proposed according to the gravity center of population instead of using the minimum and maximum boundary of all candidate solutions in a population. Park *et al.* [14] introduced a stochastic OBL based on beta distribution with partial dimensional change and selection switching into ODE to enhance the convergence speed and search ability. Pramanik *et al.* [15] presented an evolutionary technique to solve the alignment multiple molecular sequences, which used opposition based differential evolution technique with a new chromosome representation. Choi *et al.* [16] proposed a new variant of DE algorithm by combining a self-adaptive DE algorithm called dynNP-DE with Elite Opposition-Based Learning (EOBL) scheme.

It is true that computing the opposite solution at the same time when evaluating a solution of a given problem will produce another chance to look for a solution closer to the global optimum. But that's not enough. Because it is very likely that the points which are even closer to the global optimal solution in the neighborhood of the opposite point will be ignored and is not enough to increase population diversity. Hence, it is necessary to broaden the search neighborhood of the opposite candidate solutions to increase the possibility of seeking out a better solution.

To overcome the problems mentioned above, a new neighborhood opposition-based differential evolution algorithm with Gaussian perturbation (GODE) is proposed. The main idea is to search in the small neighborhood of the opposite point to enlarge the opposite area. The new OBL strategy will explore the opposite point based neighborhood after the opposite point of each individual is perturbed according to a Gaussian distribution with a self-adjusting variance. In this way, the search will decentralize on small neighborhood of the opposite point instead of a rigorous opposite point only. For the sake of verifying the performance of GODE, it is verified on CEC2014 benchmark functions [17] with 9 functions.

The remainder of this paper is organized as follows. Section II gives a preliminary knowledge, including a brief description of classical DE and an introduction of OBL. The proposed algorithm GODE is presented in Section III. The simulation experimental results and analysis are elaborated in Section IV. Finally, the work is concluded in Section V.

2 Preliminary Knowledge

2.1 Differential Evolution (DE)

Differential evolution (DE) is a population-based and directed search method [1]. Similar with other EAs, it starts with an initial population vector, which is randomly generated when no preliminary knowledge about the solution space is available. Let us assume that $X_{i,G}(i=1,2,...N_p)$ are solution vectors in generation G, where N_p is the population size. For classical DE (DE/rand/1/bin), the generation of new individuals in the population is performed by means of specified mutation, crossover, and selection operators. The three operations are directly defined as follows.

Mutation- For each vector $X_{i,G}$ in generation G, a mutant vector $V_{i,G}$ is generated by

$$V_{i,G} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G}) \quad (1)$$

where i=$\{1,2,...N_p\}$ and r_1, r_2 and r_3 are mutually different random integer indices selected from $\{1,2,...,N_p\}$. $F \in [0,2]$ is a real constant.

Crossover- DE employs a crossover operator to generate trial vectors by combining the mutant vector and the variables of the target vector, and also to increase the population diversity. The trial vector is defined as follows:

$$U_{i,G} = (U_{1i,G}, U_{2i,G}, ..., U_{Di,G}) \tag{2}$$

where $j = 1,2,...D$ (D is problem dimension) and

$$U_{ji,G} = \begin{cases} V_{ji,G}, & if\ rand_j(0,1) \le CR\ or\ j = k \\ X_{ji,G}, & otherwise \end{cases} \tag{3}$$

where $CR \in (0,1)$ is the predefined crossover probability, and $rand_j(0,1)$ is a uniform random number in [0,1], and $k \in \{1,2,...,D\}$ is a uniform random integer. The utilization of k ensures that the trial vector takes at least one component from the mutant vector.

Selection- It is to decide which vector ($U_{i,G}$ or $X_{i,G}$) should be a member of next (new) generation G+1 based on their fitness.

2.2 Opposition-Based Learning (OBL)

In some optimization problems, opposition-based learning (OBL) has been proven to be an effective concept in terms of convergence speed and solution accuracy. It aims to provide another opportunity to seek a solution closer to the global optimum. For example, let us suppose the current solution x is –20, and the global optimum is 30, then the opposite solution x^* is 20 within the range of [–100, 100]. It is easy to know that the distance of x from the global optimum is 50, whereas the distance of x^* from the global optimum is only 10. Obviously, the opposite solution x^* is closer to the global optimum. An opposition-based number can be defined as follows:

Definition (Opposite Number [12])—Let $x \in [a,b]$ be a real number. The opposite number of x^* is defined by

$$x^* = a + b - x \tag{4}$$

Similarly, the definition is popularized and applied to D-dimensional space as follows.

Definition (Opposite Point [12])—Let $X = (x_1, x_2, ..., x_D)$ be a point in D-dimensional space, where $x_1, x_2, ..., x_D \in R$ and $x_j \in [a_j, b_j]$, $\forall j \in \{1,2,...,D\}$ .The opposite point $X^* = (x_1^*, x_2^*, ..., x_D^*)$ is defined by its components

$$x_j^* = a_j + b_j - x_j \tag{5}$$

Through the above definition, the opposition-based optimization is described as follows.

Opposition-based Optimization [12]—Let $X = (x_1, x_2, ..., x_D)$ be a point in D-dimensional space (i.e., a candidate solution). Assume $f(X)$ is a fitness function which is employed to evaluate the candidate's fitness. $X^* = (x_1^*, x_2^*, ..., x_D^*)$ is the opposite point of $X = (x_1, x_2, ..., x_D)$. If $f(X^*)$ is better than $f(X)$, point X^* will substitute for point X. Otherwise, the current point X remains unchanged. Therefore, both the current point and its opposite point are evaluated and the fitter one will enter next generation.

2.3 Dynamic Opposition

Instead of using variables' predefined static interval boundaries $[a_j, b_j]$, a dynamic interval boundaries $[a_j(t), b_j(t)]$ is used to calculate the opposite of each variable based on the minimum and maximum values of that component in the current population. The dynamic opposition [20] is defined as follows:

$$OP_{ij} = a_j(t) + b_j(t) - P_{ij} \tag{6}$$

$$a_j(t) = \min(x_{ij}(t)), b_j(t) = \max(x_{ij}(t))$$

$$i = 1, 2, ..., N_p; \quad j = 1, 2, ..., D. \tag{7}$$

where P_{ij} is the *j*-th component of the *i*-th individual in the population; OP_{ij} is the opposite individual of P_{ij}; $a_j(t)$ and $b_j(t)$ are the minimum and maximum values of the *j*-th component in current population respectively, and *t* is the generation variable.

3 Proposed Algorithm: GODE

As a population-based optimization algorithm, two main steps are distinguishable for ODE, namely, population initialization and generating new individuals relying on mutation, crossover and selection operations. Both steps are enhanced by applying a neighborhood-based Gaussian perturbation to the opposite solution of each individual in the current population. The original ODE is selected as a parent algorithm and the proposed ideas are embedded in ODE to expand the search neighborhood of the opposite point and increase the likelihood of finding a better solution.

3.1 Opposition with Gaussian Perturbation

The neighborhood-based opposition search with Gaussian perturbation is given by:

$$OP_{ij}^{'} = N(x_j^*, \sigma \cdot r) \tag{8}$$

where $N(\cdot)$ is a Gaussian distribution; x_j^* is the mean value, *i.e.* the opposite of x_j; σ is the variance; *r* is a random number to adjust the perturbation radius.

For Gaussian distribution, the smaller the variance is, the larger the probability that the value of the random variable will be concentrated near x_j^*. In order to control the dispersion degree of the value of x_j^* preferably, the variance is modified as follows:

$$\sigma = \frac{b_j(t) - a_j(t)}{rand_j(N_p)} \tag{9}$$

where $rand_j(N_p)$ is a random integer between 1 and N_p. With the progress of search, the difference between $b_j(t)$ and $a_j(t)$ gradually decreases, so this is a search mode that adjusts the search step size adaptively. What's more, the using of $rand_j(N_p)$ in the variance model Eq. (9) can reduce the exploration scope for solutions to some extent.

3.2 Neighborhood Opposition-Based Population Initialization with Gaussian perturbation

On the basis of the opposition-based population initialization, Gaussian perturbation is used for each component of each individual in the opposite population OP_0 of the initial population P_0. Then the new population OP_0^1 is obtained. Algorithm1 shows the pseudocode for this phase.

3.3 Neighborhood Opposition-Based Generation Jumping with Gaussian perturbation

On the basis of the opposition-based generation jumping, Gaussian perturbation is used for each component of each individual in the opposite population OP of the current population P. Then the new population OP^1 is obtained. The pseudocode of this phase is described as algorithm 2.

Algorithm 1

/*Generate initial population P_0 */

1. **for** (i = 1 ; i < =N_p; i++)
2. **for**(j =1 ; j <= D ; j++)
3. $OP_{0i,j} = a_j + b_j - P_{0i,j}$;
4. Generate $OP_{0i,j}^1$ with Eq.(8)、(9);
5. Select N_p fittest individuals from the $P_0 \cup OP_0^1$ as initial population P_0;

Algorithm 2

/*Generation jumping stage*/

1. **If**(rand(0,1) < J_r)
2. **for** (i = 1; i <= N_p; i++)
3. **for**(j =1 ; j < =D ; j++)
4. $OP_{i,j} = a_j(t) + b_j(t) - P_{i,j}$;
5. Generate $OP_{i,j}^1$ with Eq.(8)、(9);
6. Select N_p fittest individuals from the $P \cup OP^1$ as current population P^*;

4 Numerical Experiment and Analysis

4.1 Benchmark Functions

In order to verify the performance of the proposed GODE, 9 functions of CEC2014 benchmark suite are used for simulation comparison and analysis, which are briefly divided into two categories according to their attributes： unimodal functions $(f_1 \sim f_3)$, simple multimodal functions $(f_8 \sim f_{13})$. The search ranges are all $[-100,100]^D$. More details about these functions can be found in [17]

4.2 Parameter Settings

To fairly compare the performance of DE, ODE and GODE, the same parameter settings are adopted according to the suggestions [17]. The maximum function evaluations number T is used as the stop criterion by all algorithms, which is set to 10000D. Population size N_p is set to 100. The population dimension D is 50. Differential amplification factor F is set to 0.5. Crossover probability constant C_r is set to 0.9. Jumping rate J_r is set to 0.3. The most commonly used mutation strategy DE/rand/1/bin is used. Perturbation radius parameter r = 0.001. The final results achieved by running each function independently for 30 trials in each algorithm.

4.3 Experimental Comparison

The final results achieved by DE, ODE and GODE on the 9 benchmark test functions are shown in Table I, where "Mean " indicates the mean function error value, and "Std Dev" stands for the standard deviation. As seen from Table I, for the unimodal functions $(f_1 \sim f_3)$, GODE is significantly better than DE and ODE on f_1 and f_2, and has the similar performance on f_3 with DE and ODE. Regarding the simple multimodal functions $(f_8 \sim f_{13})$, ODE is slightly superior to DE and GODE on f_8. And all three algorithms obtain the same performance on f_{13}. For the rest functions, GODE is significantly better than DE and ODE.

TABLE I. Experimental results of DE, ODE and GODE Algorithms on 9 Test Functions

Functions	DE		ODE		GODE	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
f_1	1.4955E+06	5.6005E+05	1.3907E+06	5.2385E+05	**1.2960E+06**	5.1239E+05
f_2	3.2632E+02	4.4835E+02	3.9939E+02	5.8913E+02	**2.2232E+02**	5.3050E+01
f_3	**3.0013E+02**	2.3284E-01	3.0025E+02	5.5237E-01	3.0032E+02	7.2346E-01
f_8	1.0982E+03	2.7618E+00	**1.0976E+03**	2.7384E+00	1.0987E+03	2.7123E+00
f_9	1.2548E+03	1.2191E+01	1.1391E+03	1.1555E+02	**1.0513E+03**	1.2663E+02
f_{10}	9.9877E+03	1.4630E+03	6.2424E+03	1.4340E+03	**6.0829E+03**	1.4574E+03
f_{11}	1.4149E+04	3.7072E+02	9.0700E+03	2.8473E+03	**8.7807E+03**	2.4811E+03
f_{12}	1.2032E+03	2.5637E-01	1.2017E+03	6.0633E-01	**1.2015E+03**	4.7277E-01
f_{13}	**1.3005E+03**	6.0976E-02	**1.3005E+03**	4.6086E-02	**1.3005E+03**	5.3982E-02

Furthermore, in order to statistically compare the performance differences among GODE and the compared algorithms, the Friedman test is conducted. The mean ranks of three algorithms achieved by the Friedman test are presented in Table II. From the results, it is obvious that GODE obtains the best mean ranks among all the compared algorithms. Therefore, the difference among GODE and the other two algorithms is remarkable, and the performance of GODE is remarkably better than DE and ODE.

Table II. Mean rank comparison based on Friedman test

Algorithms	Mean rank
DE	5.67
ODE	5.22
GODE	**4.11**

4.4 The Convergence Curves of the Average Best Fitness

Due to the page limit, we selected 6 different types of representative functions from the test suite and demonstrate their convergence curves of the average best fitness. These 6 functions are briefly divided into two groups: f_1 and f_2 are unimodal functions, f_9, f_{10}, f_{11} and f_{12} are simple multimodal functions. The convergence curves are plotted in Fig.1. It can be seen that the convergence speed of GODE is significantly faster than DE and ODE on f_9, f_{10} and f_{12} throughout the evolutionary process. Regarding f_1, f_2 and f_{11}, there are still some competitiveness between GODE and ODE in the middle and earlier evolutionary stage, however, GODE converges faster than ODE in later stage of the evolution and the discrepancy between them is remarkable. Overall, the performance of GODE is evidently better than DE and ODE.

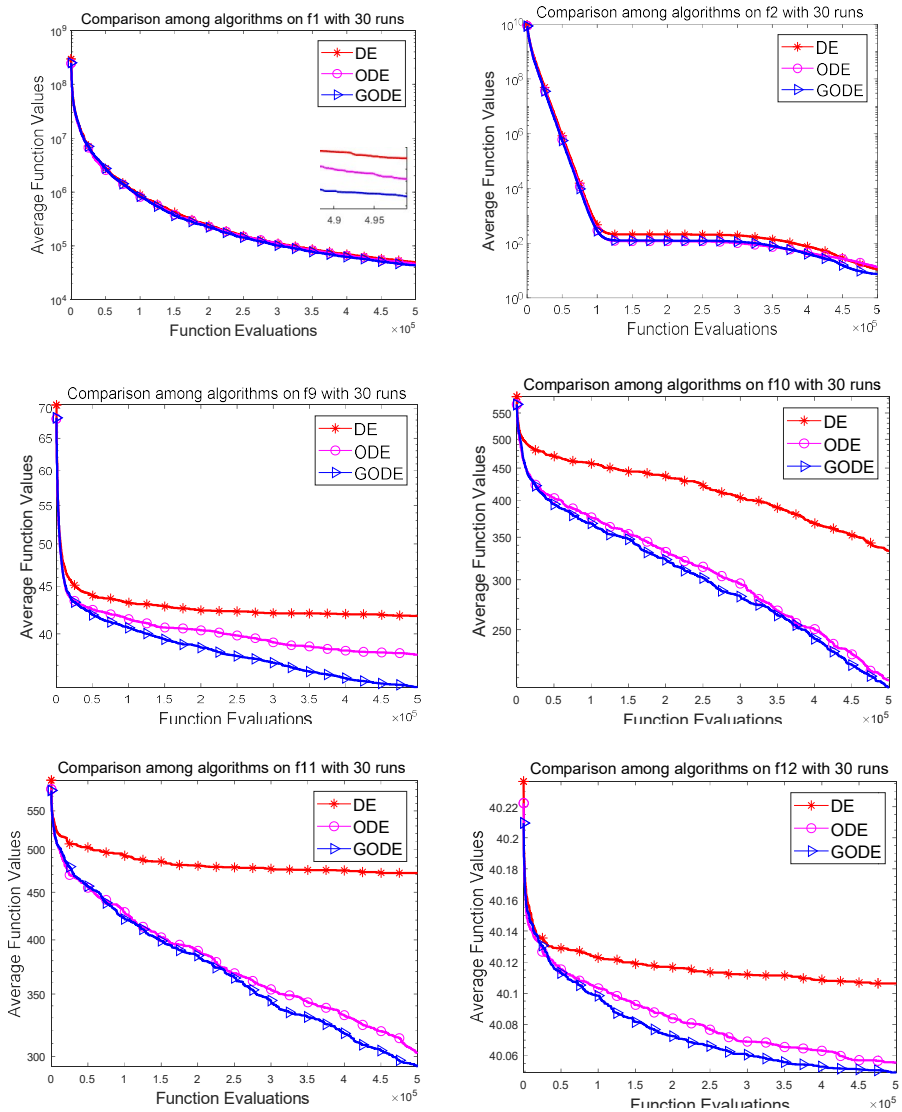


Fig.1 The convergence curves of f_1, f_2, f_9, f_{10}, f_{11} and f_{12}

5 Conclusion

This paper presents an enhanced ODE variant, called GODE, which implements Gaussian perturbation on the opposite solution of each individual in the current population. The employment of Gaussian perturbation could not only expand the search neighborhood of the opposite point but also increase the possibility of finding a better solution. The comparative experiments are conducted on 9 functions of CEC2014 test suite. Through the comparisons of experimental results and non-parametric statistical tests, it demonstrates that the performance of GODE is significantly better than DE and ODE. Therefore, GODE is an effective improvement to ODE. In future, we will further study GODE in addressing various constrained and multi-objective optimization problems.

Acknowledgements This research is partially supported by National Natural Science Foundation of China (61973042, 71772060). We will express our awfully thanks to the Swarm Intelligence Research Team of BeiYou University for the cooperative discussions.

References

1. R. Storn, K. Price, Differential evolution--A simple and efficient heuristic for global optimization over continuous spaces, J. Global Opt., vol. 11, no. 4, pp. 341-359 (1997)
2. Y. Song, B.-Y. Ma, W. Gao, S.-P. Fan, Medical Image Edge Detection Based on Improved Differential Evolution Algorithm and Prewitt Operator, Acta Microscopica., vol. 28, no. 1, pp. 30-39 (2019)
3. T.M. Li, R.-X. Yuan, X.-T. Deng, Improved self-adaptive differential evolution algorithm for reactive power optimization of smart distribution network with wind energy, International Transactions on Electrical Energy Systems., vol. 26, no. 12, pp. 2744-2758 (2016)
4. E. Hancer, Differential evolution for feature selection: a fuzzy wrapper–filter approach, Soft Computing., vol. 23, no. 13, pp. 5233-5248 (2019)
5. H.-M. Jiang, C. K. Kwong, C. Y. Chan, K. L. Yung, "A multi-objective evolutionary approach for fuzzy regression analysis," Expert Systems with Applications., vol. 130, pp. 225-235 (2019)
6. H. Xu, W.-H. Zeng, X.-X. Zeng, G.G. Yen, An Evolutionary Algorithm Based on Minkowski Distance for Many-Objective Optimization, IEEE Trans. Cybern., vol. 49, no. 11, pp. 3968-3979 (2019)
7. S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution- an update survey, Swarm Evol. Comput., vol. 27, pp. 1-30 (2016)
8. H.R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in Proc. Int. Conf. Comput. Intell. Modeling Control and Autom., Vienna, Austria, vol. 1, pp. 695-701 (2005)
9. F.M. Pour, A.A. Gharaveisi, Opposition-Based discrete action reinforcement learning automata algorithm case study: optimal design of a PID controller, Turk. J. Electr. Eng. Comput. Sci. vol. 21, no. 6, pp. 1603-1614 (2013)
10. T.J. Choi, J.H, Lee, H.Y. Youn, et al, Adaptive Differential Evolution with Elite Opposition-Based Learning and its Application to Training Artificial Neural Networks, Fundamenta Informaticae., vol. 164, no. 2-3, pp. 227-242 (2019)
11. G. T. Reddy, N. Khare, An Efficient System for Heart Disease Prediction using Hybrid OFBAT with Rule-Based Fuzzy Logic Model, Journal of Circuits. Systems and Computers., vol. 26, no. 04 (2016)
12. S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE, IEEE Trans. Evol. Comput., vol. 12, no. 1, pp. 64-79 (2008)
13. S. Rahnamayan, J. Jesuthasan, F. Bourennani, H. Salehinejad, G.F. Naterer, Computing opposition by involving entire population, in Evolutionary Computation (CEC), 2014 IEEE Congress on, pp. 1800–1807. IEEE, (2014)
14. S. Y. Park, J. J. Lee, Stochastic opposition-based learning using a beta distribution in differential evolution, IEEE Trans. Cybern., vol. 46, no. 10, pp. 2184-2194 (2016)
15. S. Pramanik, S. K. Setua, An opposition based differential evolution to solve multiple sequence alignment, Communications in Computer and Information Science., vol. 776, pp. 440-450 (2017)
16. T.J. Choi, J.H, Lee, H.Y. Youn, C.W. Ahn, Adaptive Differential Evolution with Elite Opposition-Based Learning and its Application to Training Artificial Neural Networks, Fundamenta Informaticae., vol. 164, no. 2-3, pp. 227-242 (2019)
17. J. J. Liang, B.Y. Qu, P.N. Suganthan, Problem definitions and evaluation criteria for CEC 2014 special session and competition on single objective real-parameter numerical optimization, Nanyang Technological University (Singapore) and Zhengzhou University (China)., (2013)

MISTA 2019

A HEURISTIC ALGORITHM TO COMPUTE A SUBGRAPH FOR TSP BASED ON FREQUENCY QUADRILATERALS

Yong Wang • Qirong Deng

Abstract We present a heuristic algorithm to compute a subgraph for symmetric traveling salesman problem based on frequency quadrilaterals. When we choose N frequency quadrilaterals containing an edge to compute its frequency, the frequency of an edge in the optimal Hamiltonian cycle approaches $5N$ as n is big enough. A heuristic algorithm is designed to compute a frequency graph with $N\binom{n}{2}$ randomly selected frequency quadrilaterals. A subgraph containing the optimal solution is derived according to a given frequency threshold. The algorithm requires $O(Nn^2)$ computation time, where N is much less than n. The experimental results illustrated the number of edges in complete graph is reduced by more than 30 times for certain Euclidean TSP instances of scale above 1000. Moreover, the bigger the value N is, the bigger the frequency threshold is and the smaller the number of edges in the subgraphs will be.

Keywords: Traveling salesman problem; frequency graph; frequency quadrilateral; heuristic algorithm; subgraph.

1 Introduction

TSP is one of well-known NP-hard problems in combinatorial optimization. Given a set of n vertices $\{v_1, v_2, \cdots, v_n\}$ in complete graph K_n, there is a distance $d(u, v) > 0$ for each pair of vertices $u, v \in \{v_1, v_2, \cdots, v_n\}$ and $u \neq v$. For symmetric TSP, $d(u, v) = d(v, u)$. The objective of TSP is to find an optimal Hamiltonian cycle (OHC) which is a permutation $\boldsymbol{\sigma} = (v_1', v_2', \cdots, v_n')$ of the n vertices $v_i' \in \{v_1, v_2, \cdots, v_n\}$ $(1 \leq i \leq n)$ where the distance $d(\boldsymbol{\sigma}) = d(v_1', v_n') + \sum_{i=1}^{n-1} d(v_i', v_{i+1}')$ is the minimum. In real-world applications, the vehicle routing problem, machine scheduling, very-large-scale-integration design, manufacturing path

Yong Wang
School of Renewable Energy, North China Electric Power University, No.2, Huilongguan, Changping, Beijing, 102206, China
Email: wangyyong100@163.com

Qirong Deng
School of Renewable Energy, North China Electric Power University, No.2, Huilongguan, Changping, Beijing, 102206, China
Email: 15210225792@163.com

optimization, etc., can be reduced to various TSP. Thus, TSP is widely studied in OR and computer science in order to find the efficient methods [1].

Data mining aims to find the useful and significant knowledge from data sets [2]. Many structural and semi-structural datum are usually represented as graphs. In recent years, subgraph mining becomes popular as a branch of data mining for cultivating useful subgraphs from a given large graph or set of graphs [3]. The frequent subgraph mining is concerned by researchers who regard the frequent subgraphs occurring above a frequency threshold in a given graph set [4]. The subgraphs usually have special properties which benefit the science research and engineering applications. Topological relationships between vertices are broadly considered to mine useful frequent graphs.

This research focuses on computing a subgraph for TSP on K_n. That is, given complete graph K_n of a TSP instance, we will compute a subgraph with a small number of edges for TSP. The topic is interesting because TSP on K_n and subgraphs have different complexities. In 1962, Held and Karp [5], and independently Bellman [6] designed the dynamic programing algorithm to resolve TSP in $O(n^2 2^n)$ time. Until 2010, this time was improved to $O(1.657^n)$ by Björklund [7] with an improved Monte Carlo algorithm. On the other hand, the computation time of TSP on the spanning subgraphs of K_n has been reduced to some extent. For example, TSP on cubic connected graphs can be resolved in $O(1.2312^n)$ time owing to Xiao and Nagamochi [8]. Even though the base of exponential time function of algorithms for TSP is lowered a bit, it will be referred to by most NP-complete problems. Thus, the difference of computation time of algorithms for TSP on K_n and sparse graphs promotes us to compute subgraphs for TSP.

The objective of TSP is to find an OHC within K_n. Our work is to reduce K_n of the original TSP to a subgraph containing a smaller number of edges. As the frequent subgraph mining, the subgraphs close to OHC should have a high frequency whereas the other subgraphs do not. In K_n, many subgraphs have the same topological structure. If we enumerate the subgraphs based on K_n, the number of subgraphs is so huge that the subgraphs close to OHC are hard to figure out. In this research, we first construct a set of subgraphs close to the subgraphs in OHC. Then, we enumerate the frequencies of the OHC subgraphs from the constructed subgraphs. This idea has been studied by several researchers. In computer graph algorithms, Blum [9] proposed the method to compute the backbone of a complex graph. Boese [10] employed the backbone computation for TSP by computing the intersected edges among a set of approximations. Until now, there are a few methods to compute the backbone edges for TSP. Zhang and Looks [11] computed the frequencies of edges in approximate solutions and took the edges with big frequencies as the candidate edges in OHC. Recently, Ren, Jiang, Xuan and Luo [12] used an accelerated-limit-crossing heuristics for finding the backbones for *p*-median problem and TSP. The authors believe the methods for resolving NP-hard problems will be greatly improved if the proper backbones are found.

The backbones for TSP depend on the quality of approximate solutions. Generally, it will consume much computation resources for finding the good approximations. Different from the previous research, we define a kind of optimal *k*-vertex paths instead of approximations for finding the TSP backbones. The optimal *k*-vertex paths can be computed in $O(n^k)$ time. In simplicity, we compute the optimal *k*-vertex paths containing k vertices ($k \geq 4$) in K_n and enumerate the frequencies of edges from these optimal *k*-vertex paths. The graphs computed with the optimal *k*-vertex paths are named as frequency graphs. The frequency graphs have the same topological structures as common graphs. In a frequency graph, the frequency of each edge represents its occurrence count in the optimal *k*-vertex paths. In the experiments for real-world TSP instances for k=4, the frequencies of the OHC edges are much bigger than those of most of the other edges [13]. In this paper, we proceed to compute subgraphs for TSP based on frequency quadrilaterals. According to the binomial distribution [15], the number of edges with frequency above a frequency threshold is estimated in this paper. Furthermore, the selection of frequency quadrilaterals is improved for computing the high frequency for the OHC edges. A heuristic algorithm is designed to cut many edges out of the OHC for computing a subgraph for TSP.

The rest of the paper is organized as follows. In section 2, the frequency graph for TSP is introduced briefly. In section 3, we shall review frequency quadrilaterals computed with optimal

4-vertex paths and a probability model based on frequency quadrilaterals. The possible number of edges in the preserved subgraphs is estimated in section 4. In section 5, the heuristic algorithm is designed to compute subgraphs from a frequency graph. In section 6, we do experiments to test the heuristic algorithm with the standard TSP instances in TSPLIB. The conclusions are drawn in the last section as well as the possible future research.

2 Frequency graph for TSP

We can extract all kinds of subgraphs from K_n. Moreover, we can count the frequencies of the concerned subgraphs from the extracted subgraphs. If the extracted subgraphs have some special properties, the concerned subgraphs also have the special properties. Furthermore, the subgraphs with a big frequency will intensively show the special properties of the extracted subgraphs. For example, we extract $\binom{n}{i}$ *i*-vertex complete subgraphs K_i from K_n and compute the minimum spanning tree for each of the subgraphs. After we enumerate the frequency of edges from the $\binom{n}{i}$ minimum spanning subtrees, it is obvious the edges of the minimum spanning tree of K_n have the maximum frequency.

OHC is one of spanning subgraphs of K_n. As we compute the high frequencies of the subgraphs in OHC, we will rely on the subgraphs close to the subgraphs in OHC. Two close or similar subgraphs will have the following characters: (1) They have the similar topological structure. (2) They have the similar or same special properties. We analyze the characters of the subgraphs in OHC: (1) They are *k*-vertex paths where $k \geq 2$, and (2) each *k*-vertex path is the shortest whatever the intermediate vertices are exchanged. For the first character, we may find the *k*-vertex paths in K_n as the initial subgraphs. To meet the second character, we will compute a kind of special *k*-vertex paths with two given endpoints whose distance is the minimum. The basic idea to compute special *k*-vertex paths close to the subgraphs in OHC is given as follows.

Given a set of k vertices $\{v_1, v_2, \cdots, v_k\}$ in K_n where $k \geq 4$, let $\boldsymbol{v} = (v'_1, v'_2, \cdots, v'_k)$ be a permutation of the k vertices where $v'_1 = v_1$ and $v'_k = v_k$. The distance of $\boldsymbol{v}$ is computed as $d(\boldsymbol{v}) = \sum_{i=1}^{k-1} d(v'_i, v'_{i+1})$. We assume all the paths $\boldsymbol{v}$s have different distances. There is one path $\boldsymbol{v}$ whose distance $d(\boldsymbol{v})$ is the minimum. We call this path the optimal *k*-vertex path for end vertices v_1, v_k. We have $\binom{k}{2}$ ways to choose the end vertices from $\{v_1, v_2, \cdots, v_k\}$ so that there are $\binom{k}{2}$ optimal *k*-vertex paths.

Now, we find the optimal *k*-vertex paths which have the second property as that of the paths in OHC. The next work is to compute the optimal *k*-vertex paths and enumerate the frequency of the *i*-vertex paths where $i < k$. Since the optimal *k*-vertex paths have the same characters as the paths in OHC, the *i*-vertex paths in OHC will have high frequencies after they are enumerated from the optimal *k*-vertex paths.

In K_n, there are $\binom{n}{k}$ *k*-vertex sets. Thus, there are total $\binom{k}{2}\binom{n}{k}$ optimal *k*-vertex paths. As *n* is fixed, the number of optimal *k*-vertex paths changes as a combinatorial number according to *k*. Wang [14] has computed the total number of optimal *k*-vertex paths in K_n as $\binom{n}{2}2^{n-2}$. It is still hard to find an OHC through traversing all optimal *k*-vertex paths.

One possible method is to compute the frequencies of optimal *i*-vertex paths with optimal *k*-vertex paths where $i < k$. The optimal *i*-vertex paths with big frequencies are taken as the candidate paths in OHC. Before the computation, the values of *k* and *i* should be determined. The time complexity to compute the optimal *k*-vertex paths is $O(n^k)$. In general, the bigger the *k* is, the closer the optimal *k*-vertex paths approach the paths in OHC. We will obtain the outstanding frequencies for the *i*-vertex paths in OHC according to the optimal *k*-vertex paths where *k* is big. On the other hand, it will consume much time. In theory, Section 3 shows the *i*-vertex paths in OHC will have the biggest frequency as *n* is big enough and *k* is bigger than 3. To reduce the computation time, we compute the optimal 4-vertex paths and count the frequencies of the optimal *i*-vertex paths ($i < 4$). For small or medium scale TSP instances, the frequencies of all OHC *i*-vertex paths will not be outstanding. The merits are also attractive since we will save much computation time. Moreover, the frequencies of most *i*-vertex paths in OHC

will be very high. The optimal 4-vertex paths are composed of 3-edge or 2-edge paths (i=3 and 2). For convenience, we enumerate the frequencies of edges from the optimal 4-vertex paths.

Given a set of optimal 4-vertex paths, the frequency of each edge is counted. The edges and their frequencies from a frequency graph. In such frequency graphs, the frequency of the OHC edges is generally much bigger than that of most of the other edges. In a previous paper [14], we computed the frequency graphs with optimal 4-vertx paths and filter out many useless edges according to a given frequency threshold. The time complexity of the previous algorithm is $O(n^4)$ which is difficult to apply to large scale TSP. In this paper, we compute the frequency graphs with randomly selected frequency quadrilaterals. Furthermore, we design a heuristic algorithm which runs in $O(Nn^2)$ time to compute a subgraph with a small number of edges for TSP, where N is much smaller than n.

3 The probability model based on frequency quadrilaterals

3.1 Frequency quadrilaterals

The frequency quadrilaterals have been introduced in a recent paper [15]. Given four vertices $\{A, B, C, D\}$ in K_n, they form a quadrilateral ABCD. ABCD includes six optimal 4-vertex paths. Due to the simplicity of quadrilateral, the optimal 4-vertex paths are computed with a four-vertex and three-line inequality [16]. Then, the corresponding frequency quadrilateral is computed with the six optimal 4-vertex paths. We give the method to compute a frequency quadrilateral as follows.

A weighted quadrilateral ABCD is shown in Figure 1. The distances of edges d_{AB}, d_{AC}, d_{AD}, d_{BC}, d_{BD} and d_{CD} are noted aside the edges where $d_{AB}=d(A,B)$, $d_{AC}=d(A,C)$, $d_{AD}=d(A,D)$, $d_{BC}=d(B,C)$, $d_{BD}=d(B,D)$ and $d_{CD}=d(C,D)$.

It is the three sum distances related to the three pairs of opposite edges $d_{AB}+d_{CD}$, $d_{AC}+d_{BD}$ and $d_{AD}+d_{BC}$ that determine the six optimal 4-vertex paths in ABCD. We assume the three sum distances $d_{AB}+d_{CD}$, $d_{AC}+d_{BD}$ and $d_{AD}+d_{BC}$ are unequal for computing the six optimal 4-vertex paths. Since there are three sum distances, they naturally have six different permutations and each permutation constitutes an inequality array. The six inequality arrays are given in Table 1. Each independent inequality array is used to compute six optimal 4-vertex paths in ABCD. Then, the six optimal 4-vertex paths are used to compute the corresponding frequency quadrilateral. The six frequency quadrilaterals are shown in Figure 2. Given an arbitrary weighted quadrilateral ABCD in K_n, the corresponding frequency quadrilateral will be one of the six frequency quadrilaterals in Figure 2.

Let's see the frequencies of edges in the six frequency quadrilaterals. For an edge $e \in \{AB, AC, AD, BC, BD, CD\}$, the frequency f of e is 5, 3 and 1, respectively. In addition, each of the frequencies 5, 3 and 1 occurs twice in the six frequency quadrilaterals, respectively. For $i \in \{1,3,5\}$, let $p_i(e)$ be the probability that e has frequency i in a frequency quadrilateral containing e. Thus, the probability $p_1(e) = p_3(e) = p_5(e) = \frac{1}{3}$ according to the six frequency quadrilaterals. In K_n, each ABCD has the equal probability to be one of the six frequency quadrilaterals in Figure 2. Given N frequency quadrilaterals containing e, there will be $\left[\frac{N}{3}\right]$ frequency quadrilaterals where e has frequency f = 1, 3 and 5, respectively.

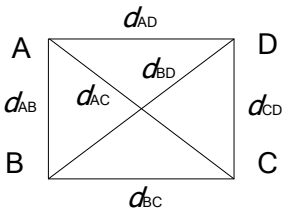


Fig. 1 A weighted quadrilateral ABCD

Table 1. The six inequality arrays and the corresponding optimal paths

No.	Inequality array	Six optimal 4-vertex paths
(1)	$d_{AB}+d_{CD}<d_{AD}+d_{BC}<d_{AC}+d_{BD}$	(A,D,C,B), (A,B,D,C), (A,B,C,D), (B,A,D,C), (B, A,C,D), (C,B,A,D).
(2)	$d_{AB}+d_{CD}<d_{AC}+d_{BD}<d_{AD}+d_{BC}$	(A,C,D,B), (A,B,D,C), (A,B,C,D), (B,A,D,C), (B, A,C,D), (C,A,B,D).
(3)	$d_{AD}+d_{BC}<d_{AB}+d_{CD}<d_{AC}+d_{BD}$	(A,D,C,B), (A,D,B,C), (A,B,C,D), (B,A,D,C), (B, C,A,D), (C,B,A,D).
(4)	$d_{AD}+d_{BC}<d_{AC}+d_{BD}<d_{AB}+d_{CD}$	(A,D,C,B), (A,D,B,C), (A,C,B,D), (B,D,A,C), (B, C,A,D), (C,B,A,D).
(5)	$d_{AC}+d_{BD}<d_{AB}+d_{CD}<d_{AD}+d_{BC}$	(A,C,D,B), (A,B,D,C), (A,C,B,D), (B,D,A,C), (B, A,C,D), (C,A,B,D).
(6)	$d_{AC}+d_{BD}<d_{AD}+d_{BC}<d_{AB}+d_{CD}$	(A,C,D,B), (A,D,B,C), (A,C,B,D), (B,D,A,C), (B, C,A,D), (C,A,B,D).

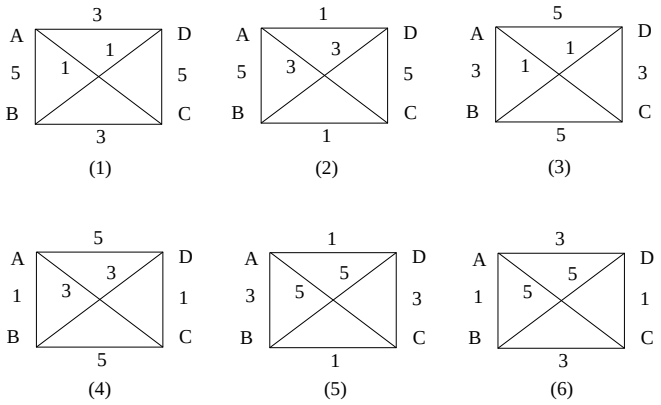


Fig. 2 Six frequency quadrilaterals for a weighted quadrilateral ABCD.

3.2 A theorem based on frequency quadrilaterals

We use the frequency quadrilaterals instead of optimal 4-vertex paths for computing the frequencies of edges. Given *N* frequency quadrilaterals containing edge *e*, the frequency of *e* is computed as $(e)=\sum_{i=1}^{N} f_i$, where f_i is the frequency *f* of *e* in the i^{th} frequency quadrilateral. Since frequency quadrilaterals are computed with optimal 4-vertex paths, the frequencies of edges computed with frequency quadrilaterals has the same performance as those computed with the optimal 4-vertex paths. Differently, we can estimate $F(e)$ of the OHC edges with a binomial distribution model based on frequency quadrilaterals [15].

In K_n, edge e is contained in $\binom{n-2}{2}$ quadrilaterals so there are the same number of frequency quadrilaterals containing *e*. When we choose *N* frequency quadrilaterals containing *e*

to compute $F(e)$, $F(e) = 3N$ according to the probability $p_1(e) = p_3(e) = p_5(e) = \frac{1}{3}$. This is the average frequency for all of edges. For an OHC edge *e*, Wang and Remmel [15] found some special frequency quadrilaterals where the frequency of *e* is 5 or 3 rather than 1. They gave the probability model (1) for the OHC edges, where they combined $p_3(e)$ and $p_5(e)$ together as $p_{3,5}(e) = \frac{2}{3} + \frac{2}{3(n-2)}$.

$$\begin{cases} p_1(e) = \dfrac{1}{3} - \dfrac{2}{3(n-2)} \\ p_{3,5}(e) = \dfrac{2}{3} + \dfrac{2}{3(n-2)} \end{cases} \quad (1)$$

When we use *N* frequency quadrilaterals containing an OHC edge *e* to compute its frequency, $F(e) = \left(3 + \frac{2}{n-2}\right) N$ which is bigger than 3*N*. They did experiments for certain TSP instances. It illustrated that the probability model (1) was too conservative for the OHC edges. The minimum frequency of the OHC edges is much bigger than $\left(3 + \frac{2}{n-2}\right) N$ for most TSP instances. If we use $\left(3 + \frac{2}{n-2}\right) N$ as the frequency threshold to trim the edges with frequency $F(e) < \left(3 + \frac{2}{n-2}\right) N$, we will compute a subgraph with less than $\frac{1}{2}\binom{n}{2}$ edges. To compute a subgraph as sparse as possible, we expect the probability $p_{3,5}(e)$ for OHC edges is as big as possible so that we can use a big frequency threshold *F* to eliminate more edges.

According to the probability model (1), we know $p_{3,5}(e) > \frac{2}{3}$ for an OHC edge. Thus, we assume $p_{3,5}(e) = \frac{2}{3} + \epsilon$ for an OHC edge where $\epsilon > 0$. We are interested in the change of $p_{3,5}(e) = \frac{2}{3} + \epsilon$ according to the scale of TSP. In paper [17], Wang proven the following theorem for cultivating $p_{3,5}(e)$ for an OHC edge according to *n*.

Theorem 1*: If $p_{3,5}(e) = \frac{2}{3} + \epsilon$ is the minimum probability of an OHC edge in a frequency quadrilateral in K_n, $p_{3,5}(e) = \frac{2}{3} + \epsilon$ tends to* 1 *as n is big enough.*

For an OHC edge, the minimum probability $p_{3,5}(e) = \frac{2}{3} + \epsilon$ increases according to *n* until it reaches the maximum value 1. In this case, we assume *e* has the probability $p_3(e) = p_5(e) = \frac{1}{2}$ in each of the *N* frequency quadrilaterals. The minimum frequency of an OHC edge computed with *N* frequency quadrilaterals will be equal to 4*N*. According to the similar proof process, we can prove $p_5(e) \to 1$ for an OHC edge as *n* is big enough under the assumption of $p_5(e) = \frac{1}{3} + \epsilon$ and $\epsilon > 0$. In the ultimate case, the frequency $F(e)$ of each OHC edge tends to 5*N* as *n* is sufficiently large. For large scale of TSP instances, we can improve the frequency threshold to eliminate more edges with a lower frequency.

If every quadrilateral includes exactly six optimal 4-vertex paths, an OHC edges will have the probability $p_{3,5}(e) \to 1$ or $p_5(e) \to 1$ for big number *n*. The OHC edges will have frequencies 3, 5 in most of their frequency quadrilaterals. When we choose *N* frequency quadrilaterals containing *e* to compute *F*(*e*), *F*(*e*) will be bigger than 4*N* and tend to 5*N* as *n* is big enough. On the other hand, the frequencies of most of the other edges will be smaller than 4*N*. In view of the frequencies of edges, we can eliminate the edges according to the frequency threshold 4*N* or close to 5*N* and preserve a subgraph containing OHC for TSP.

4 The possible number of edges in a subgraph

The next questions are how big the frequency threshold is and how many edges the subgraph contains if we use a given frequency threshold. In practice, the scale of most TSP instances is limitative. $p_{3,5}(e)$ and $p_5(e)$ of some OHC edges will be less than 1. In this case, *F* close to 5*N* will be too big as the frequency threshold.

F can be estimated according to the probability $p_{3,5}(e) = \frac{2}{3} + \epsilon$ and $p_1(e) = \frac{1}{3} - \epsilon$. We assume $p_3(e) = p_5(e) = \frac{1}{3} + \frac{\epsilon}{2}$ so the frequency threshold is equal to $F = 3(1 + \epsilon)N$. *F* depends on the parameter ϵ. The bigger the scale *n* is, the bigger the ϵ is as well as the frequency threshold *F*. When ϵ is small, there will be the other edges whose probability $p_{3,5}(e)$ is equal to or above $\frac{2}{3} + \epsilon$. We assume there are *K* edges with probability $p_{3,5}(e) \geq \frac{2}{3} + \epsilon$ in K_n. In average case, an edge *e* has the probability $p_1(e) = p_3(e) = p_5(e) = \frac{1}{3}$ in a given frequency quadrilateral containing *e*. As we choose *N* frequency quadrilaterals containing *e*, the probability that there are $\left[\left(\frac{2}{3} + \epsilon\right)N\right]$ frequency quadrilaterals where *e* has frequency 3 or 5 is computed as $\binom{N}{\left(\frac{2}{3}+\epsilon\right)N}\left(\frac{2}{3}\right)^{\left(\frac{2}{3}+\epsilon\right)N}\left(\frac{1}{3}\right)^{\left(\frac{1}{3}-\epsilon\right)N}$. It is a value of the binomial distribution $P(X = i) = \binom{N}{i}\left(\frac{2}{3}\right)^{i}\left(\frac{1}{3}\right)^{N-i}$ where *X* is the random variable denoting the number of frequency quadrilaterals where *e* has frequency 3 or 5. We will preserve the edges with a probability $p_{3,5}(e) \geq \frac{2}{3} + \epsilon$. The accumulative probability is $P\left(X \geq \left[\left(\frac{2}{3} + \epsilon\right)N\right]\right) = \sum_{i=\left[\left(\frac{2}{3}+\epsilon\right)N\right]}^{N}\binom{N}{i}\left(\frac{2}{3}\right)^{i}\left(\frac{1}{3}\right)^{N-i}$ where$i \geq \left[\left(\frac{2}{3} + \epsilon\right)N\right]$. Considering the $\binom{n}{2}$ edges, the number *K* of preserved edges will be the formula (3) as $F = 3(1 + \epsilon)N$ is used as the frequency threshold.

$$K = \binom{n}{2}\sum_{i=\left[\left(\frac{2}{3}+\epsilon\right)N\right]}^{N}\binom{N}{i}\left(\frac{2}{3}\right)^{i}\left(\frac{1}{3}\right)^{N-i} \quad (2)$$

The binomial distribution $P(X = i) = \binom{N}{i}\left(\frac{2}{3}\right)^{i}\left(\frac{1}{3}\right)^{N-i}$ has the maximum probability at $i = \left\lceil\frac{2}{3}(N+1)\right\rceil - 1$ or $i = \left\lfloor\frac{2}{3}(N+1)\right\rfloor$. After that, it decreases exponentially from $i = \left\lceil\frac{2}{3}(N+1)\right\rceil$ or $i = \left\lfloor\frac{2}{3}(N+1)\right\rfloor + 1$ to *N*. Once $\epsilon > \frac{1}{N}$, $\left[\left(\frac{2}{3} + \epsilon\right)N\right] > \left[\frac{2}{3}(N+1)\right]$ or $\left[\left(\frac{2}{3} + \epsilon\right)N\right] > \left\lfloor\frac{2}{3}(N+1)\right\rfloor + 1$, the value $\binom{N}{\left(\frac{2}{3}+\epsilon\right)N}\left(\frac{2}{3}\right)^{\left(\frac{2}{3}+\epsilon\right)N}\left(\frac{1}{3}\right)^{\left(\frac{1}{3}-\epsilon\right)N}$ deviates from the maximum probability and tens to zero quickly. Even though ϵ takes the conservative value $\frac{2}{3(n-2)}$ as that in formula (1), *K* will be very small when $N = nlogn$ or a bigger value. In theory, we can compute the subgraphs with a small number of edges for TSP if the OHC edges have $p_{3,5}(e) \geq \frac{2}{3} + \epsilon$ based on frequency quadrilaterals. It is granted especially for big scale of TSP.

K depends on the parameter ϵ. The bigger the ϵ is, the smaller the *K* is. For the edges with $p_{3,5}(e) < \frac{2}{3} + \epsilon$ (or $p_5(e) < \frac{1}{3} + \epsilon$), there is a small probability for them to have $\left[\left(\frac{2}{3} + \epsilon\right)N\right]$ (or $\left[\left(\frac{1}{3} + \epsilon\right)N\right]$) frequency quadrilaterals where their frequency is 3 or 5 (or only 5) among *N* frequency quadrilaterals. These edges will be trimmed according to frequency threshold. For the other edges with $p_{3,5}(e) \geq \frac{2}{3} + \epsilon$ (or $p_5(e) \geq \frac{1}{3} + \epsilon$), there is a big probability that they have $\left[\left(\frac{2}{3} + \epsilon\right)N\right]$ (or $\left[\left(\frac{1}{3} + \epsilon\right)N\right]$) frequency quadrilaterals where their frequency is 3 or 5 (or only 5). However, the number of these edges is very small according to formula (2). Most edges will have approximate $\left[\frac{2}{3}N\right]$ (or $\left[\frac{1}{3}N\right]$) frequency quadrilaterals where their frequency is 3 or 5 (or only 5) among *N* frequency quadrilaterals. These edges will be neglected. Thus, we will preserve a small number *K* of edges in the subgraphs.

Through experiments, we can approximately determine the parameter values of *N* and ϵ for various TSP instances. In the next section, we will design a heuristic algorithm to compute a subgraph for TSP based on the Theorem (1).

5 The heuristic algorithm

To compute a subgraph for TSP based on frequency quadrilaterals, some techniques need to tackle. For an OHC edge, $p_{3,5}(e) \to 1$ or $p_5(e) \to 1$ as n is big enough. In real-world applications, TSP scale is generally not so big. Therefore, $p_{3,5}(e)$ or $p_5(e)$ of some OHC edges will be less than 1, especially for small and medium scale of TSP instances. When we select N frequency quadrilaterals at random for an edge e, $F(e)$ of some OHC edges will deviate from $4N$ or $5N$. To compute the a big $F(e)$ for the OHC edges, we should choose the frequency quadrilaterals where their frequency is 5 or 3. In a frequency quadrilateral, the two opposite edges associating with the minimum sum distance have the frequency 5 (see section 2.1). Here we give an applicable method to choose such frequency quadrilaterals for the OHC edges. For an edge e with distance $d(e)$, we choose the non-adjacent edges g whose distance meets the constraints $d(g) < cd$. Parameter c is a small value, such as c<1.5. c can be adjusted according to the TSP instances. Two non-adjacent edges e and g compose a quadrilateral. We restrict the distance $d(g)$ of g in the N quadrilaterals in order to compute a big frequency for e in the corresponding frequency quadrilaterals containing e and g.

Every edge is included in $\binom{n-2}{2}$ quadrilaterals in K_n. Given edge e, we have to order the $\binom{n-2}{2}$ opposite edges according to their distances for choosing N suitable quadrilaterals. The distances sorting for $\binom{n}{2}$ edges will consume $O(n^4 \log n)$ time. To save computation time, we use another way to choose the opposite edges of e for constructing N quadrilaterals. We first order the $\binom{n}{2}$ edges according to their distances from small to big values and an edge sequence $\left(e_1, e_2, \cdots, e_{\binom{n}{2}}\right)$ is obtained. The first edge e_1 has the minimum distance and the last edge $e_{\binom{n}{2}}$ is the longest. For an edge $e_i \in \left(e_1, e_2, \cdots, e_{\binom{n}{2}}\right)$ where $1 \le i \le \binom{n}{2}$, we choose its opposite edges g from the edge subsequence $(e_1, e_2, \cdots, e_{i+m})$ where m is a given number and $i + m \le \binom{n}{2}$. With this method, we only order the $\binom{n}{2}$ edges once and it needs $O(n^2 \log n)$ time if the quick sorting algorithm is adopted. The parameter m is close to i to ensure that the distance $d(g)$ is not far bigger than $d(e_i)$. In addition, the previous several shortest edges e_i are always maintained, such as i<10. This treatment does not affect the computation time and results much.

The other question is how many frequency quadrilaterals we should choose for every edge e_i. If N is too small, the method will lose the power of the probability model and binomial distribution model. The number K computed with formula (3) will be big as we use a small frequency threshold F. Otherwise, we will lose some OHC edges in the subgraphs if F is too big. When a big number of frequency quadrilaterals are used, the OHC edges will be preserved even though big frequency threshold is adopted. However, the algorithm will last long computation time. The suitable parameter N can be determined according to experimental results. We suggest N<n in order to reduce the computation time.

Since we choose the quadrilaterals for e where $d(e) + d(g)$ is small, the frequency of e will be 5 or 3 in most of the N corresponding frequency quadrilaterals if e has a big probability $p_{3,5}$ or p_5 in K_n. Otherwise, e will not have many such frequency quadrilaterals. To simplify the computation, we compute $F(e)$ with the frequency quadrilaterals where e has frequency f=5. The frequency threshold F is assigned a value near to $5N$. The experiments in the next section showed that $F(e)$ of most of OHC edges is close to $5N$ for the TSP instances in TSPLIB [18].

Given a TSP instance on n vertices, $\binom{n}{2}$ edges are ordered first to form an edge sequence $\boldsymbol{s} = \left(e_1, e_2, \cdots, e_{\binom{n}{2}}\right)$. We use the quick sorting algorithm and it needs $O(n^2 \log n)$ time. The heuristic algorithm to compute subgraphs is given in Figure 3.

At the first step, the prepared datum is input. They are the vertex set, edges' distances in K_n and the ordered edge sequence $\left(e_1, e_2, \cdots, e_{\binom{n}{2}}\right)$. The three parameters N, m and F are also assigned at the first step. The following steps are the kernel computation process of the heuristic algorithm. For an edge $e_i \left(1 \le i \le \binom{n}{2}\right)$, it first chooses N non-adjacent edges g of e_i from the

edge subsequence $(e_1, e_2, \cdots, e_{i+m})$ where $i + m \leq \binom{n}{2}$. m is small to prove the distance constraint $d(g) < cd(e_i)$. Each edge g and e_i form one quadrilateral. For the short edges e_i, the sum distance $d(e_i) + d(g)$ in each of the N quadrilaterals is small. Thus, e_i will have the frequency 5 in most of the N frequency quadrilaterals. On the other hand, this method cannot guarantee the long edges e_i to have frequency 5 in most of the N frequency quadrilaterals. After N quadrilaterals containing e_i and g are determined, they are converted into the corresponding frequency quadrilaterals. Then, $F(e_i)$ is computed with the frequency quadrilaterals where e_i has frequency 5. For the OHC edges, they will have frequency 5 in most of the N frequency quadrilaterals. After $F(e_i)$ is calculated, we compare $F(e_i)$ with F. If $F(e_i)>F$, e_i is preserved in the subgraph. Otherwise, e_i is eliminated. The computation process runs until all of edges are traversed. The final step is to output the subgraph.

Algorithm 1. The process to compute a subgraph

Input: The edge sequence $\left(e_1, e_2, \cdots, e_{\binom{n}{2}}\right)$ according to their distances from small to big values.
The vertex set and distances of edges in K_n.
Parameters m, N and F.
index i=0.
Repeat until $\boldsymbol{i = \binom{n}{2}}$
For $\boldsymbol{e_i}$, index k = 0.
Repeat until $\boldsymbol{k=N}$
Choose an edge g from the edge subsequence $(\boldsymbol{e_1, e_2, \cdots, e_{i+m}})$ where $\boldsymbol{i + m \leq \binom{n}{2}}$.
if (g is opposite to $\boldsymbol{e_i}$ in one quadrilateral) k++.
End
Convert the N quadrilaterals into the corresponding frequency quadrilaterals.
Compute the $\boldsymbol{F(e_i)}$ with the frequency quadrilaterals where $\boldsymbol{e_i}$ has frequency f=5.
if ($\boldsymbol{F(e_i)} > F$) keep $\boldsymbol{e_i}$.
Else eliminate $\boldsymbol{e_i}$.
i++.
End
Output: The subgraph with K edges.

Fig. 3 The heuristic algorithm to compute a subgraph for TSP

The computation time of the algorithm relies on two parameters: the scale n of TSP and the number N. In the kernel computation process, there are two loops. In the outer loop, the index i goes from 1 to $\binom{n}{2}$ for every edge in the edge sequence. In the inner loop, the index k goes from 1 to N to choose N non-adjacent edges g for each edge. Thus, the heuristic algorithm requires $O(Nn^2)$ computation time. The parameter N can be adjusted to prove the accuracy and efficiency of the algorithm. In current stage, we suggest $N \ll n$ so the computation time of the heuristic algorithm is smaller than $O(n^3)$.

There are three parameters m, N and F in the algorithm. To save the computation time, N should be as small as possible. However, the probability model does not work well as N is too small, for example N<30. In general, the value span [100, 1000] is a good choice for moderate and large scale of TSP. A small value of m is useful to compute the high frequency for the OHC edges. It should not be too small for choosing N opposite edges for some very short edges. In applications, m can be assigned a value close to the value of N. A big value of F is useful to cut many edges of the lower frequency. According to Formula (1) and Theorem 1, an OHC edge will have a bigger frequency than a common edge. Plus the selection of frequency quadrilaterals for the OHC edges, the value of F will tend to 5N. Therefore, F can be assigned a value above 4.9N and it will increase according to n.

6 Experiment and analysis

In this section, we shall do experiments with TSP instances to test the heuristic algorithm. The TSP instances are selected from TSPLIB. The heuristic algorithm is run on a laptop with a 2.3GHz CPU and 4.0GB inner memory. Before we execute the heuristic algorithm, OHCs of these TSP instances are computed with Concorde online [19]. The OHC is used to verify whether

the preserved subgraphs contain OHC or how many OHC edges are lost in the subgraphs under the assignments of the three parameters *N*, *m* and *F*.

There is one problem we should explain before the experiments. For some TSP instances, they have many equal-weight edges. Given a quadrilateral ABCD, the three sum distances $d_{AB} + d_{CD}, d_{AD} + d_{BC}, d_{AC} + d_{BD}$ will usually equal. In such cases, it is hard to compute the six optimal 4-vertex paths and the exact frequency quadrilateral. To compute one frequency quadrilateral for such ABCD, we add small random distances $rd \in [0,1]$ to the edges' distances to make the three sum distances $d_{AB} + d_{CD}, d_{AD} + d_{BC}, d_{AC} + d_{BD}$ as unequal as possible. The function of small random distance *rd* has been used in paper [15]. Due to the randomness of *rd*, some frequency quadrilaterals cannot be rightly computed with respect to OHC in one experiment. In probability, it will compute the useful frequency quadrilaterals in most experiments.

We first assign the three parameters *N*, *m* and *F* for the heuristic algorithm. For each TSP instance, the parameters may be different to compute the best subgraph due to the scale *n* and different structures of TSP. We will choose the proper values for the three parameters for every TSP instance to compute the subgraphs.

In the first round of our experiment, we take *m*=100. *N* and *F* are not fixed. For each TSP instance, we first assign the number *N* and then the frequency threshold *F* is tried. For the big TSP instances, we use the relatively big values of *N* and *F*. We tried the frequency threshold *F* to compute the subgraphs containing OHC or the subgraphs only lose a few OHC edges. Four types of TSP instances are used to verify the heuristic algorithm. For different types of TSP, they usually have different inherent structures. According to the preserved subgraphs, we can estimate the performance of the heuristic algorithm for different types of TSP. Since we add small random distances to edges' distances, the results in two or more experiments will be different. Thus, we give three experimental results for each TSP instances. The experimental results are shown in Table 2. Under the assignments of parameters *N*, *m* and *F*, we compute three subgraphs. *K* denotes the number of edges. *l* is the number of lost OHC edges in each subgraph, which is behind *K*. *l*=0 is not shown in the table if the subgraph contains OHC. The average degree $\bar{d}$ of vertices in the three subgraphs is computed according to the three numbers *K*. The average computation time $\bar{t}$ of the three experiments is also given for each TSP instance.

Table 2. The experimental results with fixed parameters *N*, *c* and *F*

TSP	Size *n*	*N*	*m*	*F*	Edges number *K*/*l*			$\bar{d}$	$\bar{t}$/s
a280	280	100	100	4.9*N*	1815	1800	1846	13	0.3
gr431	431	600	100	4.9*N*	5818/4	5826/3	5863/4	27	3.5
att532	532	300	100	4.9*N*	7988	7974/1	7949/1	30	2.9
u574	574	100	100	4.9*N*	7420	7371	7329	26	1.1
rat783	783	200	100	4.92*N*	9997	10039	9893	25	4.5
pr1002	1002	200	100	4.93*N*	11508/1	11604/1	11521/1	23	7.8
si1032	1032	200	100	4.94*N*	12226/2	12047/1	12104/3	23	8.1
pcb1173	1173	200	100	4.94*N*	17470	17565	17513	30	11.3
d1655	1655	200	100	4.94*N*	45864/1	45961	45743	55	27
u1817	1817	200	100	4.94*N*	54555	54778	54882	60	33.7
u2152	2152	250	100	4.95*N*	64520/1	64288	64562	60	61.7
pr2392	2392	250	100	4.95*N*	94586	94760	94078/1	79	79.9
pcb3038	3038	300	100	4.95*N*	148365/1	148402/1	148379/3	98	164.1
rl5934	5934	500	100	4.97*N*	569592/1	570698/1	570049	192	1277.8

With the assignments of *N*, *m* and *F*, the heuristic algorithm computes the subgraphs for all the TSP instances. For each TSP instance, the subgraphs usually contain OHC and sometimes they lose a few OHC edges in the experiments. The probability that the subgraphs contain OHC is very high. It validates the theory in Sections 3 and 4. Many useless edges are trimmed from K_n for all these TSP instances. In view of *K*, the number of edges is reduced by more than 20 times for all the Euclidean TSP instances. Moreover, the number of edges is reduced by more than 30 times for the Euclidean TSP instances of big scale above 783. We also see the frequency threshold $F \geq 4.9N$ for these TSP instances. It means each of the OHC edges has the frequency 5 in most of the *N* selected frequency quadrilaterals. Thus, the OHC edges have a big probability $p_{3,5}(e) = \frac{2}{3} + \epsilon$ (or $p_5(e) = \frac{1}{3} + \epsilon$) in a given frequency quadrilateral. In addition, *F* increases according to the scale of TSP. For the other kinds of TSP instances, such as GEO (gr431) and ATT (att532) TSP, the number of edges in the subgraphs is reduced by more than 15 times. However, for the MATRIX TSP (si1032), the number of edges in the subgraphs is reduced by more than 44 times. It says the heuristic algorithm is useful to compute the subgraphs not only for Euclidean TSP instances, but also for the other types of TSP instances.

In the paper [20], an iterative algorithm is used to eliminate the edges based on frequency quadrilaterals. In each iteration, the half edges are cut according to the frequencies of edges. In the experiment, the iterative algorithm requires $O(n^4)$ time and it is hard to apply to the big size of TSP instances. Comparing with the previous algorithms based on frequency quadrilaterals, the heuristic algorithm eliminates more edges at one step. Moreover, it requires $O(n^2)$ time.

It mentions that there are many quadrilaterals ABCD where $d_{AB} + d_{CD}, d_{AD} + d_{BC}, d_{AC} + d_{BD}$ are equal for some of these TSP instances, especially for ATT, GEO and MATRIX TSP instances. The small random distances *rd* cannot guarantee to compute the biggest frequency for the OHC edges in every experiment. If we did more experiments for these examples, the better subgraphs will be computed.

Table 3. The experimental results with various *N* and *F* for rat783 where *m* = 100

F \ *N*	50	100	150	200	250	300	350	400	450	500
4.9*N*	18712	14344	12981	12258	11789	11453	11298	11167	11053	10979
4.91*N*	9772/14	9905/4	9966	9940	10109	10038	10002	10007	9976	9221
4.92*N*	9818/11	9788	9959	9939	9998	8516/1	8798	8960	9052	9130
4.93*N*	9902/10	9978/3	9951/1	7712	8321	8459	7477/1	7735	8040	7382
4.94*N*	9867/8	9922/2	6951/3	7691/1	8184	7015	7430	6630	7043	7296
4.95*N*	9832/13	5402/21	7021/5	5485/4	6469/4	5492/1	6188	5529	5988	5562/2
4.96*N*	9882/10	5405/21	7005/3	5554/5	6429/1	5523/5	4828/5	5495/2	5019/2	5612
4.97*N*	9811/7	5345/24	3867/23	5489/7	4595/10	3965/17	4833/4	4345/6	3994/4	3620/9
4.98*N*	9759/12	5438/20	3920/29	3132/41	4619/10	3946/7	3458/15	3209/21	2913/27	3689/5
4.99*N*	9918/9	5453/19	3912/39	3159/30	2714/39	2384/51	2157/57	1970/52	1834/55	1752/73

To show the change of results according to N and F, we did the second round of experiments for some TSP instances. In this experiment, we change the parameters N and F to compute subgraphs for rat783. The parameter m=100. The number K of edges and the number l of the lost OHC edges in the subgraphs are illustrated in Table 3. N changes from 50 to 500 and the difference between each two adjacent Ns is 50. Given a specific N, the frequency threshold varies from 4.9N to 4.99N and the interval is 0.01N. For a fixed N, the bigger the frequency threshold F is, the smaller the number K is. Meanwhile, the more number of OHC edges will be lost in the subgraphs as F becomes bigger. If we use a small number N, the method will lose the power of the probability model and binomial distribution model. As N is smaller, such as N=50, 100, etc., the smaller F is suitable to compute a subgraph with more number of edges containing OHC. As N rises, we can use the relatively bigger frequency threshold F so that the sparser

subgraphs are computed. Thus, the big value of N is a good choice to compute the sparse subgraphs. The disadvantage is that the algorithm will consume more computation time.

For a specific value of *F*, the bigger the *N* is, the smaller the number of edges in the subgraphs is. For example, the subgraphs computed according to *N*=500 is generally sparser than that computed according to *N*<500 if the same *F* is used. It means a big *N* value is helpful to compute a sparse subgraph for TSP. This reason can be explained with formula (3). As *N* becomes bigger, the probability $\binom{N}{\left(\frac{2}{3}+\epsilon\right)N}\left(\frac{2}{3}\right)^{\left(\frac{2}{3}+\epsilon\right)N}\left(\frac{1}{3}\right)^{\left(\frac{1}{3}-\epsilon\right)N}$ will become smaller for a given parameter ϵ since $\left\lceil\left(\frac{2}{3}+\epsilon\right)N\right\rceil$ deviates from $\left\lceil\frac{2}{3}(N+1)\right\rceil-1$ or $\left\lfloor\frac{2}{3}(N+1)\right\rfloor$ further. Thus, *K* computed with formula (3) decreases according to *N*. In the heuristic algorithm, the number of edges with frequency $F(e_i) > F$ becomes less as *N* rises. In addition, the OHC edges will have more number of the frequency quadrilaterals where their frequency is 5 due to their big probability $p_{3,5}$ or p_5. Thus, *F* is improved and the subgraph with a small number of edges is computed. In general, *F* will approach 5*N* based on the big value of *N* and small value of *m*. However, it will consume much computation time if *N* is very big.

m is important to choose the frequency quadrilaterals for an edge where it has the frequency 5. When *m* is small, the distances *d*(*g*) are less than *d*(*e*) in most of the selected quadrilaterals containing *e* and *g*. If *d*(*e*)+*d*(*g*) is very small, *e* will have the frequency 5 in most of these corresponding frequency quadrilaterals. Thus, *F*(*e*) will approach 5*N*. On the other hand, if *m* is too big, we will have a big probability to obtain *d*(*g*)>*d*(*e*) in the constructed quadrilaterals. In this case, *d*(*e*)+*d*(*g*) may be bigger than the sum distances of the other two pairs of opposite edges. There will be more number of frequency quadrilaterals where *e* has the small frequency 1 and 3. In such cases, a small *F* should be considered to compute a subgraph containing OHC but with more number of edges.

We did the third round of experiment for rat783 to see the preserved subgraphs according to changes of *m* and *N*. *F* is assigned as 4.96*N* and *N* still changes from 50 to 500. We give three values below and above 100 for *m*, respectively. The minimum value of *m* is 10 and the maximum value of *m* is 1000. The number *K* of edges and the number *l* of lost OHC edges in the subgraphs are given in Table 4.

As *N* is fixed, the number of edges becomes smaller as *m* rises. It means we construct less percent of the frequency quadrilaterals where *e* has frequency 5 among the *N* frequency quadrilaterals. *F*=4.96*N* is bigger than *F*(*e*) for more number of OHC edges as *m* rises. As *N* is small (*N*=50, 100, 150, etc.), it is not suitable to use a big *m*. Otherwise, the subgraphs will lose quite a few OHC edges if we use a big frequency threshold. As *N* is big (*N*>300), the preserved graphs include nearly all of OHC edges whatever *m* changes. We can compute the subgraphs containing OHC according to a big *N* and the effect of *m* can be neglected. *F*(*e*) of the OHC edges are determined by the structure of K_n. The OHC edges have the big probability $p_{3,5}$ or p_5 in their frequency quadrilaterals. One sees *K* becomes smaller as *N* and *m* become bigger. It implies us to compute a sparse subgraph with the big parameters values of *m* and *N*. In this case, the frequency threshold *F* also rises to 5*N*.

Table 4. The experimental results with various *N* and *m* for rat783 when *F* = 4.96*N*

m \ *N*	50	100	150	200	250	300	350	400	450	500
10	9854/5	5429/13	6976/6	5480/4	6468	5507/2	4958/3	5521/1	5010/2	5599
30	9822/12	5517/13	7012/1	5504/4	6468/1	5466	4898/8	5498/2	5068/1	5506
50	9831/14	5415/19	6971/3	5506/3	6454/1	5557	4885/1	5560/1	5019/2	5572
70	9914/7	5469/13	7044/3	5491/8	6377	5463/1	4861/1	5557/2	4977/3	5584
200	9800/11	5377/21	6965/3	5449/5	6314/1	5429/3	4811/3	5467/2	4949	5465/2
500	9737/14	5275/25	6832/7	5376/4	6271/4	5346/1	4738/4	5387/1	4879	5359/1
1000	9380/21	5056/42	6672/4	5166/6	5989/3	5211/2	4548/10	5138/2	5110	4646/2

Through the experiments, *N* plays the first important role to compute a sparse subgraph containing OHC for TSP. The bigger the *N* is, the bigger the *F* will be. *m* is the second useful parameter to compute a satisfactory subgraph. The big *m* is helpful to compute a sparse subgraph as parameter *N* is big enough. As *N* is small, we generally use a small *m*. The third parameter *F* relies on *N*. The smaller the *N* is, the smaller the *F* is and the bigger the *K* is. In addition, the scale of TSP is not neglected when we assign the three parameters. In view of the results in the Table 2, 3 and 4, the three parameters *N*, *m* and *F* should be given the bigger values as *n* rises. According to the experimental results, we suggest $N<n$, $m=o(n)$ and *F* is close to 5*N*. When *m* is too big and $i+m$ approaches $\binom{n}{2}$, we will need a lot of frequency quadrilaterals for each edge to compute a sparse subgraph. If *N* is too big, we will compute the satisfactory subgraphs whereas the heuristic algorithm will consume long computation time, especially for large *n*.

7 Conclusions

This work computes a subgraph for TSP based on frequency quadrilaterals. The probability $p_{3,5}(e)$ (or $p_5(e)$) of the OHC edges is bigger than $\frac{2}{3}+\epsilon$ (or $\frac{1}{3}+\epsilon$) based on frequency quadrilaterals. As *n* is big enough, the probability $p_{3,5}(e)$ (or $p_5(e)$) of an OHC edge tends to 1. The number of edges in the subgraphs is derived as $\binom{n}{2}\sum_{i=\left\lfloor\left(\frac{2}{3}+\epsilon\right)N\right\rfloor}^{N}\binom{N}{i}\left(\frac{2}{3}\right)^{i}\left(\frac{1}{3}\right)^{N-i}$ if we use the frequency threshold $3(1+\epsilon)N$. A heuristic algorithm containing three parameters *N*, *m* and *F* is designed to compute a subgraph for TSP. The experiments show the number of edges in the subgraphs is reduced by more than 30 times for large scale of Euclidean TSP where $N \ll n$, *m* is a small number and *F* is close to 5*N*. If we use the bigger values *N* and *m*, the better results will be computed. On the other hand, the algorithm will consume more computation time. The heuristic algorithm based on the frequency quadrilaterals has much space to improve. In the future, we will study what kind of subgraphs we will compute if the procedure is iterated.

Acknowledgements The authors acknowledge the funds supported by the Fundamental Research Funds for the Central Universities (Nos. 2018ZD09 and 2018MS039) and NSFC (No.51205129). The work benefits from the facilities of National Key Laboratory of New Energy Power System and the Beijing Key Laboratory of New and Renewable Energy, North China Electric Power University, Beijing, China.

References

1. G. Gutin and A.P. Punnen, The traveling salesman problem and its variations, 489-576, Kluwer Academic Publishers, Springer US, New York (2007)
2. J. Han and M. Kamber, Data mining concepts and techniques, 3rd Edition, San Francisco: Morgan Kaufmann, America(2014)
3. C. Jiang, F. Coenen and M. Zito, A survey of frequent subgraph mining algorithms, The Knowledge Engineering Review, Vol.28, 75–105(2013)
4. D.J. Cook and L.B. Holder, Substructure discovery using minimum description length and background knowledge, Journal of Artificial Intelligence Research, 231–255(1994).
5. M. Held and R.M. Karp, A dynamic programming approach to sequencing problems, Journal of the Society for Industrial and Applied Mathematics, Vol.10, 196–210(1962)
6. R. Bellman, Dynamic programming treatment of the travelling salesman problem, Journal of the ACM, Vol.9, 61–63(1962)
7. A. BjöJrklund, Determinant sums for undirected Hamiltonicity, Computer Science, 173–182(2010)

8. M. Xiao and H. Nagamochi. An Exact Algorithm for TSP in Degree-3 Graphs Via Circuit Procedure and Amortization on Connectivity Structure, Algorithmica, Vol.74, 713–741(2016)
9. H. Blum, Models for the Perception of Speech and Visual Form, 362–380, Cambridge: M1T Press, England(1967)
10. K.D. Boese, Cost versus distance in the traveling salesman problem, Technical Report CSD-950018(1995)
11. W.X. Zhang and M. Looks, A novel local search algorithm for the traveling salesman problem that exploit backbones, Proceedings of the 19th International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, 343–350(2005)
12. Z.L. Ren, H. Jiang, J.F. Xuan, and Z.X. Luo, An Accelerated-Limit-Crossing-Based Multilevel Algorithm for the p-Median Problem, IEEE Transactions on Systems, Man, and Cybernetics - Part B:Cybernetics, Vol.42, 1187–1202(2012)
13. Y. Wang, The frequency graph for the traveling salesman problem, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering, Vol.6, 1189–1192(2012)
14. Y. Wang, An approximate method to compute a sparse graph for traveling salesman problem, Expert Systems with Applications, Vol.42, 5150–5162(2015)
15. Y. Wang and J.B. Remmel, A binomial distribution model for the traveling salesman problem based on frequency quadrilaterals, Journal of Graph Algorithms & Applications, Vol.20, 411–434(2016)
16. Y. Wang, An approximate algorithm for triangle TSP with a four-vertex-three-line inequality, International Journal of Applied Metaheuristic Computing, Vol.6, 35–46(2015)
17. Y. Wang, Sufficient and necessary conditions for an edge in the optimal Hamiltonian cycle based on frequency quadrilaterals, Journal of Optimization Theory and Applications, Vol.181, 671–683(2019)
18. G. Reinelt, TSPLIB (1995), http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
19. H. Mittelmann, NEOS Server for Concorde (2015), http://neos-server.org/neos/solvers/co:concorde/TSP.html
20. Y. Wang and J.B. Remmel, An iterative algorithm to eliminate edges for traveling salesman problem based on a new binomial distribution, Applied Intelligence, Vol.48, 4470–4484 (2018)

MISTA 2019

Applying resource-leveling strategy for solving Multi-Skill Resource-Constrained Project Scheduling Problem

Yuan Tian, Tifan Xiong , Zhenyuan Liu, Pengcheng Deng, Li Wan

Abstract Resource leveling is generally considered as a non-regular indicator for scheduling problems. In this paper, we studied the possibility to apply the resource-leveling strategy to solve the multi-skill resource-constrained project scheduling problem with the criteria of minimizing the makespan and the total cost simultaneously. First, a brief introduction to this problem and its presentation is given. Next, the relationship between the resource loads and the objectives of solutions is studied. Then, this strategy is applied using the framework of Fruit Fly Optimization Algorithm. Finally, the experiment result validates the effectiveness of this strategy.

1 Introduction

Project scheduling is an important task in project management. The resource-constrained project scheduling problem (RCPSP) [1] is one of the most widely studied scheduling problems. It can be described as follows[2]. A single project which consists of real activities has to be

Yuan Tian
School of Mechanical Science and Engineering, Huazhong University of Science and Technology
E-mail: tianyuan_me@hust.edu.cn

Tifan Xiong (corresponding author)
School of Mechanical Science and Engineering, Huazhong University of Science and Technology
E-mail: xiongtf@hust.edu.cn

Zhenyuan Liu
School of Automation, Huazhong University of Science and Technology
Email: zyliu@mail.hust.edu.cn

Pengcheng Deng
School of Mechanical Science and Engineering, Huazhong University of Science and Technology
Email: 13164645450@qq.com

Li Wan
School of Mechanical Science and Engineering, Huazhong University of Science and Technology
Email: wanli@hust.edu.cn

scheduled subject to precedence and renewable and nonrenewable resource constraints such that the time required for performing all activities, the makespan, is minimized.

While the RCPSP is already a powerful model, it cannot cover all situations in practice. Multi-skill resource-constrained project scheduling problem (MS-RCPSP) is an extension[3], [4] of RCPSP. In the MS-RCPSP, each resource masters several skills with given levels, and each task in the project requires a certain skill at a standard level to be executed. This type of problem easily emerges when human resources[5], [6] or multi-purpose machines[7] are involved in a project.

As being an NP-hard problem[8], heuristic and meta-heuristic algorithms are widely adopted to obtain near-optimal solutions within a reasonable computational effort. Some researchers modified algorithms that were originally for RCPSP to solve MS-RCPSP: Genetic Algorithm[9], Differential, Ant Colony Optimization[10], Teaching-learning-based optimization[11], Fruit fly Optimization[12], etc. Generally, these methods use different strategies to explore potential solutions and preserve discovered "best" solutions, e.g. crossover[9], [11], swap operation[11], [13].

To best of our knowledge, most proposed algorithms solving MS-RCPSP were simply migrated with modified encoding and decoding procedures, schedule generation schemes and evaluators. Few of them considered approaches from the characteristics of resources and tasks of MS-RCPSP.

In this paper, we proposed a strategy from the perspective of resource-leveling[2] – a non-regular objective that measures the resource loads. Commonly, duration and cost are the main objectives for solving MS-RCPSP and we found the utilization and load of resources can indicate the position of the current solution (We consider the search behavior as a process to find Pareto Front, duration and cost are the objectives that should be evaluated.). With the guidance of this indicator, algorithms can better search large solution space in a direction and find solutions as much as possible. The goal of this paper is to investigate whether the proposed strategy can enhance the quality of algorithms in solving MS-RCPSP.

The rest of the paper is organized as follows. Section 2 shows the problem statement with constraints and requirements is described. The general procedure of algorithms solving MS-RCPSP is introduced in Section 3. In Section 4 the relationship between the quality of solutions and its resource loads is investigated, while Section 5 shows the practice and experiments of this operator. Finally, Section 6 concludes the article and presents potential directions for future work.

2 Problem Description

In this paper, we adopted the model proposed by Myszkowski[14], [15] who published a dataset called iMOPSE which is a public library for MS-RCPSP. For further information, please visit: http://imopse.ii.pwr.wroc.pl/

MS-RCPSP is composed of three parts: **tasks** T, **skills** Q and **resources** R. Each task is defined by task **duration** and requires a **skill** to be handled. There exist **precedence relations** between some tasks, which means some tasks have to be performed after some others finished. Besides, every task is **non-preemptive**.

The second part is the **resources** that are characterized by their salary and specified set of owned skills**.** A resource is capable of performing a task **if and only if** the resource involves the required skill and the familiarity level is no less than the standard required by the specific task. Each resource has a **fixed unit salary**, regardless of what skill is needed. In addition, a resource is dedicated so it cannot execute multi-tasks at the same time.

Skill is varied with its familiarity levels, denoted as an integer value from 0 to 3. The **skill** required by a task to be executed determines which resource can be assigned to it.

The objective of MS-RCPSP is to assign the resource to every task to get a feasible schedule either it is the shortest/ cheapest. Cost and duration are usually two conflicted objective(time and cost trade-off problem), thus it could be presented as a **multi-objective**

optimization problem[16]: minimization of the project schedule's performance cost and minimization of project schedule's duration.

☑can be assigned
☒cannot be assigned

			T1: 6	T2: 3	T3: 4	T4: 5
			-	-	*T2*	-
			Q3:1	Q2:1	Q1:1	Q1:0
R1:11	Q2:1	Q3:1	✓	✓	×	×
R2:16	Q1:1	Q3:2	✓	×	✓	✓
R3 : 9	Q1:1	Q2:0	×	×	✓	✓
R4:13	Q2:1	Q3:1	✓	✓	×	×

Fig. 1 Example of a project

To better explain this model, the relationship between tasks, resources and skills is shown as a matrix in Fig. 1. The first row and the first column are task ID and resource ID with its duration and salary, respectively. The task ID in italic in the second row shows the precedence relations, e,g. Task 3 has to start only after T2 is finished. As for resource capability, taking R3 as an example; it owns two skills, i.e. Q2 with level 1 and Q2 with level 0. Thus, it can perform T3 and T4 and is not suitable for T1. An example of solution is illustrated in Fig. 2, where the start time of each task and its assigned resource is given with total duration 12 and cost 126.

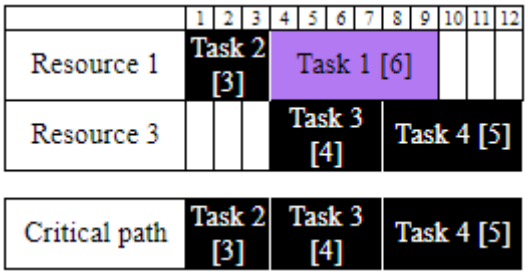


Fig. 2 Example of solution for MS-RCPSP

Mathematical formulation plays an important role [17], [18] in solving RCPSPs and models are usually classified as discrete- or continuous-time models, according to how time is look at. In this paper, the continuous-time formulation is adopted where continuous variables are used to represent the start time of the tasks involved in the project and we follow some of notations and symbols proposed by Correia and Saldanha-da-Gama. [19] to formulate the problem we are investigating.

Before deepening the study on the MS-RCPSP, the following notations are defined.

Index:

i, j The tasks i and j.

k The resource k.

l The skill l.

Sets and parameters:

$\boldsymbol{V} = \{0, 1, \dots, n+1\}$ Set of the tasks to be executed. Tasks 0 and $n+1$ are the dummy tasks, where the former represents the beginning of project and the latter

represents the end. Note that these dummy tasks are assumed to have no resource requirements and have processing time equal to 0.

$\boldsymbol{\mathcal{R}} = \{1, \dots, K\}$ Set of the resources.

$\boldsymbol{\mathcal{L}} = \{1, \dots, L\}$ Set of the skills.

$\boldsymbol{\mathcal{L}}_k$ Set of the skills owned by resource $k \in \boldsymbol{\mathcal{R}}$.

l_i the skill required by task $i \in \mathbf{V}$.

h_l Level of the skill l $\in \boldsymbol{\mathcal{L}}$.

p_i Processing time for task $i \in \mathbf{V}$.

s_k Salary of resource $k \in \boldsymbol{\mathcal{R}}$(hourly rate).

$\boldsymbol{P}_i$ Set of predecessor tasks of task $i \in \mathbf{V}$.

E Set of the pairs of tasks between which no precedence relation (direct and transitive) exists.

$\boldsymbol{\mathcal{R}}_i = \{k \in \mathcal{R} \mid \exists l' \in \boldsymbol{\mathcal{L}}_k, l' = l_i \wedge h_{l'} \geq h_{l_i}\}$ Set of the resources that are capable to perform the task $i \in \mathbf{V}$.

$\mathbf{V}_k = \{i \in \mathrm{V} \mid \exists l' \in \boldsymbol{\mathcal{L}}_k, l' = l_i \wedge h_{l'} \geq h_{l_i}\}$ Set of the tasks that can be executed by resource $k \in \boldsymbol{\mathcal{R}}$.

Decision Variables:

Considering that non-negative decision variables for the start time of the tasks and binary variables defining the execution order for any pair tasks are always used to formulate the project scheduling problem, the following decision variables are considered for formulating the studied MS-RCPSP problem:
S_i: Start time for task $i \in \mathbf{V}$.

$$x_{ik} = \begin{cases} 1, \text{ if resource k perform the task i} \\ 0, \text{ otherwise} \end{cases} \quad (i \in \mathrm{V}; k \in \boldsymbol{\mathcal{R}}_i)$$

$$y_{ij} = \begin{cases} 1, \text{ if task } j \text{ starts after task } i \text{ is completed} \\ 0, \text{ otherwise} \end{cases} \quad (i, j \in \mathrm{V} \wedge (i, j) \in \mathrm{E})$$

It should be noted that variables y_{ij} are only constrained for pairs $(i, j) \in$ E because no such decision is needed for all the other situations.

Considering the decision variables above, we can write the following model:

$$\min \tau = S_{n+1} \quad (1)$$

$$\min c=\sum_{i\in \boldsymbol{V}}\sum_{k\in \boldsymbol{\mathcal{R}}_i} x_{ik}\cdot s_k\cdot p_i \tag{2}$$

Subject to:

$$S_i + p_i \le S_j;\ \forall i,j \in \boldsymbol{V} \wedge i \in \boldsymbol{P}_j \tag{3}$$

$$S_i + p_i - M(1-y_{ij}) \le S_j;\ \forall i,j \in \boldsymbol{V} \wedge (i,j) \in \mathrm{E} \tag{4}$$

$$\sum_{k\in \boldsymbol{\mathcal{R}}_i} x_{ik} = 1;\ \forall i \in \boldsymbol{V} \tag{5}$$

$$y_{ij} + y_{ji} \le 1;\ \forall i,j \in \boldsymbol{V} \wedge (i,j) \in \mathrm{E} \tag{6}$$

$$x_{ik} + x_{jk} \le y_{ij} + y_{ji} + 1;\ \forall i,j \in \boldsymbol{V}_k \wedge (i,j) \in \mathrm{E}, \forall k \in \boldsymbol{\mathcal{R}}_i \tag{7}$$

$$S_i \ge 0;\ \forall i \in \boldsymbol{V} \tag{8}$$

$$x_{ik} \in \{0,1\};\ \forall i \in \boldsymbol{V}, \forall k \in \boldsymbol{\mathcal{R}}_i \tag{9}$$

$$y_{ij} \in \{0,1\};\ \forall i,j \in \boldsymbol{V} \wedge (i,j) \in \mathrm{E} \tag{10}$$

Equations (1) and (2) describe the two optimization objectives, i.e. minimizing makespan and the total cost, respectively. Constraint (3) ensures that all the tasks should satisfy the precedence constraint relations. In constraint (4) M denotes a number which is large enough. This constraint ensures that if y_{ij} is equal to 1, then task j can only start after task i is finished. Constraint (5) guarantees that each task must be performed by only one resource during the project scheduling. Constraint (6) shows that either task j starts after task i starts or task i starts after task j starts or none of the previous. Constraint (7) ensures that if one resource k is assigned to two tasks i and j (assuming $i,j \in V_k$ and $(i,j) \in$ E), then $y_{ij} + y_{ji} \ge 1$, which together with Constraint (6) ensures that $y_{ij} + y_{ji} = 1$, i.e. either task i finishes before task j starts or task j finishes before task i starts. It also means that each resource can only perform one task at a time. Finally, Constraints (8)-(10) state the domain of the decision variables.

3 Encoding and Decoding Scheme

The encoding and decoding scheme can be considered as the input and output of metaheuristics. The decoding scheme is also known as Schedule Generation Scheme[20](SGS) in RCPSP. These two schemes are usually interdependent. Resource assignment vector χ is indispensable, which has a length of n element equals to the number of tasks in the project, the index i could represent i-th task and its value $\chi(i)$ means the resource ID that is assigned to. For example, $\chi = 1\ 1\ 3\ 3$ means R1 is assigned to T1 and T2, R3 to T3 and T4, the resource capability and its allocation situation is the same in Fig. 1 and Fig. 2. This vector represents the assignment leaving the start time of each task not defined, that is where the SGS takes over. Similar to RCPSP, SGS could apply various priority rules[21], [22] to create feasible schedules with different objective values.

There are also algorithms adopted two-vector[13], [23] as the encoding scheme. Besides the previously described resource assignment vector, a task vector determines the order of tasks to schedule. The corresponding SGS does not need to apply rules to decide which tasks to

schedule first. However, this scheme would lead the programming complicated since the transformation of task vector, e.g. crossover and swap, may violate the precedence relations and an extra operator should be provided to repair this vector.

4 The Proposed Resource-Leveling Strategy

To work with two objective optimizations, it is necessary to define a relation of domination – one point in the solution space to dominate anther point if at least one of its objective values is better and no other objective value is worse than that of the second point. A Pareto Front (PF) is a set of points that are not dominated. True Pareto Front is unknown and the process of searching the solution space is to get a potential Pareto Front. For MS-RCPSP, decision-makers usually focus on cost and duration of the whole project. Fig. 3 illustrates a typical Pareto Front which was solved by NTGA (a variant of Genetic Algorithm for MS-RCPSP)[9], it provides a set of feasible solutions with their objectives. This instance comes from iMOPSE and its definition name is 100_10_65_15, the following discussion is based on this instance.

To best of our knowledge, most proposed metaheuristics for MS-RCPSP simply adopted general mechanisms to improve the diversity and the convergence of solutions or to preserve potential solutions, for example, swapping adjacent elements[13] of a vector or crossover vectors[11] from solutions. Few of them consider the features of this schedule itself.

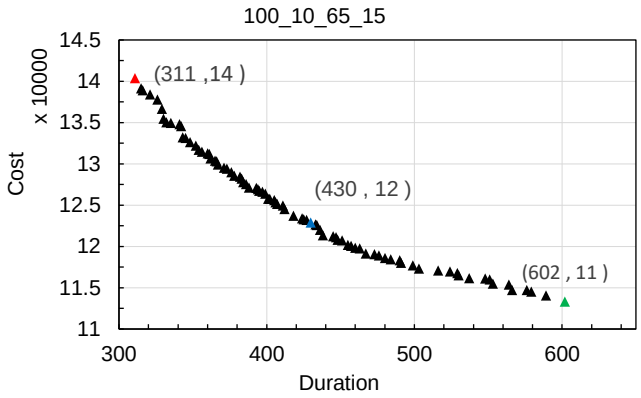


Fig. 3 A typical Pareto Front of MS-RCPSP

Resource leveling[24] is a technique to maintain a uniform profile of resource requirements in the project life span. In MS-RCPSP, resources determine not only the cost but also the duration of a whole project. Fig. 4 presents the 3 solutions from both ends and the middle point of the Pareto Front in Fig. 3. The numbers after the colon on the x-axis indicates the unit salary of this resource and resources are sorted in ascending order. We can see the cheapest solution (total duration: 602, cost:1.4×10^5) seldom uses expensive resources, i.e. R4 and R8, while most tasks are finished by R7 and R9, hence its duration is the longest one. The solution with both objectives in a medium level tends to assign relative expensive resources to speed up the duration of the project. A balanced resource assignment can be seen in the shortest solution where each resource is exploited as fully as possible. From here we may find that the process of finding solutions can be seen as a course of balancing resource loads.

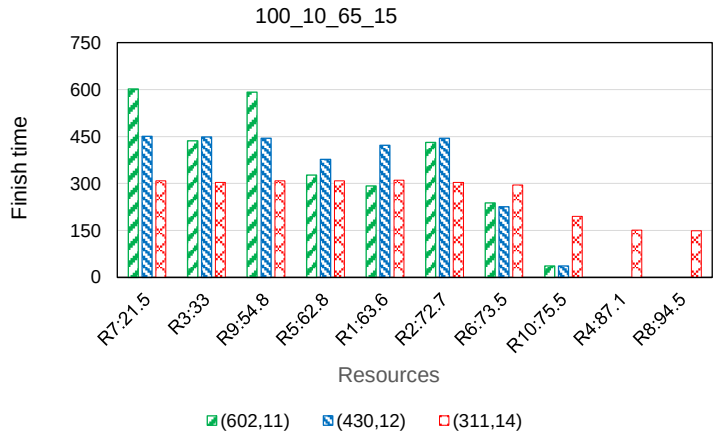


Fig. 4 Resource loads of different solutions

The degree of resource leveling seems positively correlated with the evolutionary of solutions. Fig. 5 shows how resource utilization changes with the number of iterations. Resource leveling rate is defined by the standard deviation of the finish time of all resources. (SGS has ensured the idle time of each resource as tight as possible and the latest finished time of resources is the total duration of the project). As the number of generations increases, some solutions move up which means they are closer to True Pareto Front and more solutions are discovered. At the same time, the resource utilization rate goes down with the duration decreasing, this trend proves a short schedule has a balanced resource assignment.

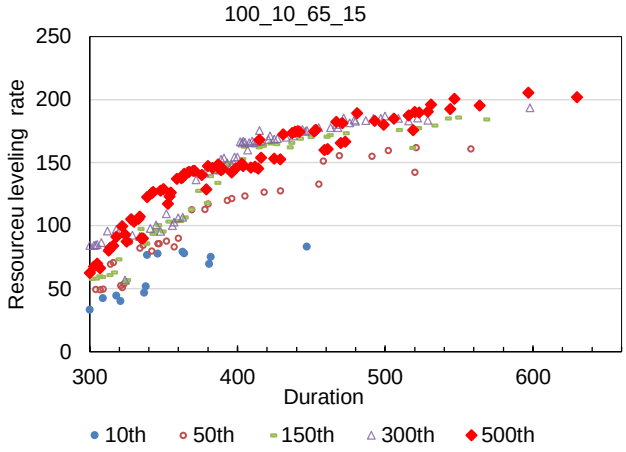


Fig. 5 Solutions improved with iterations

From the above analysis, we can see new solutions can generate by smoothing the resource workload, reassigning tasks to other “idle” resources may improve the duration of the project. That strategy could be effective than randomly mutate or swap the elements in a resource vector. Besides, due to the characteristic of this model – task duration is irrelevant to resources and its skill level, and the resource unit salary is a fixed number, the non-dominated point of each instance – the longest but cheapest solution is known by assigning tasks with the cheapest capable resource which happens to be the most unbalanced allocation. If we use that as a starting point and continuously improve its resource load distribution, the trajectory may be the potential Pareto Front.

5 Experiment and Validation

5.1 Framework of the proposed algorithm

To apply the resource leveling strategy into practice, we incorporated it into the fruit fly optimization algorithm(FOA) [12] and make it as resource-leveling fruit fly optimization algorithm(RLFOA). Naturally, we would consider the process of counterbalancing resource utilization as a process of finding a solution with the minimum resource utilization rate, the trajectory method[25] i.e. Simulating Annealing or Hill Climbing seems to be an appropriate method. Using a single solution to search may be vulnerable to get trapped in local optimal, therefore population-based optimization would be a better choice.

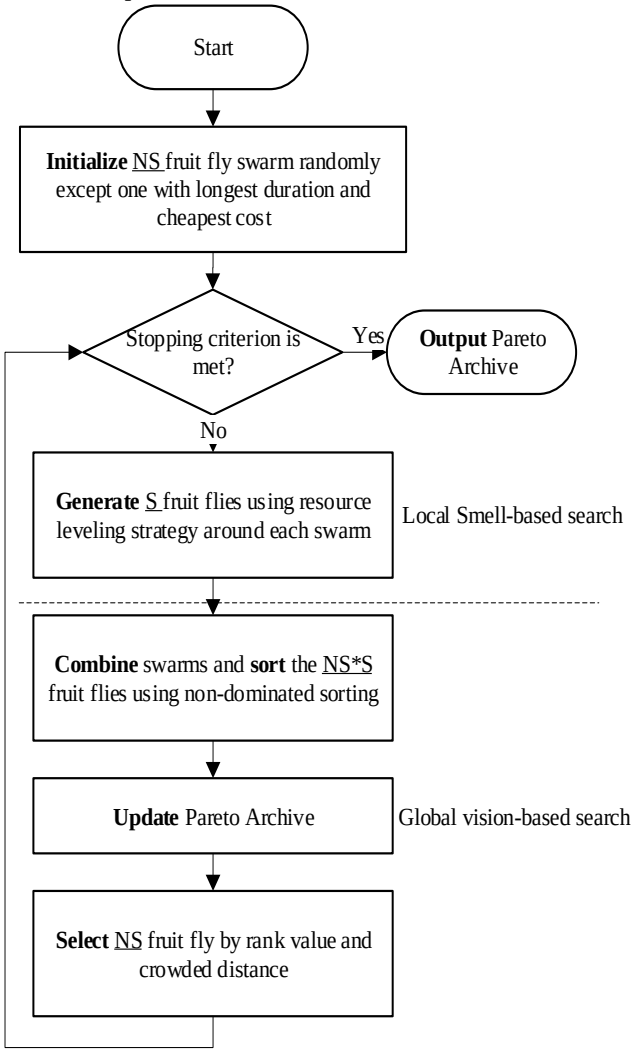


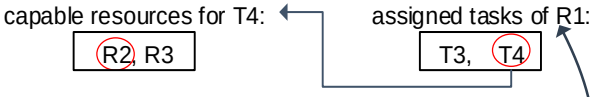
Fig. 6 Flowchart of the algorithm

The procedure of the algorithm (Fig. 6) is a simplified version of FOA as the purpose of this experiment is to validate the effectiveness of the resource-leveling strategy. This algorithm only uses the resource-leveling strategy to generate new solutions and the non-dominated sorting technique to find the Pareto Front. This strategy is implemented as follow:

Resource-leveling strategy

Step1: Evaluate the load (finish) time of each resource of the current solution, all resources are classified by its skill type-level.
Step 2: Randomly pick a group and use the roulette method to select a resource. The resource with a higher finish time is more likely to be chosen.
Step 3: Tasks assigned to this resource has a probability to reassign to another resource. Meanwhile, a new resource assignment vector is generated, and a complete solution is built by SGS.
Step 4: Check whether the modified solutions improve and could replace the existed solutions.

An example is given in Fig. 7 where the definition of problem is the same as Fig. 1. This is a typical cheapest and longest schedule since R2 and R4 are never used. Then, we randomly pick a skill type-level, assuming Q1:1 is selected. It is obvious that R3 will be chosen after the roulette method as the probability of choosing R3 equals to 1.0, while R2 equals to 0 according to their finish times. Next, find the tasks assigned to R3, randomly select one of them and reassign this task of another capable resource. We can see this schedule improves after assigning R2 to T4 and a new non-dominated solution is created.



Skill type-level	R1:11	R2:16	R3:9	R4:13
Q1:1	-	0	12	-
Q2:0	-	-	12	-
Q2:1	9	-	-	0
Q3:1	9	-	-	0
Q3:2	-	0	-	-

Fig. 7 An example of resource leveling strategy

5.2 Test data

iMOPSE[14], [15] is a public library dedicated to MS-RCPSP. It provides useful tools for researchers to test their own methods, e.g. instance Generator, Validator and Visualizer. Besides, there is a predefined dataset (36 instances) and best-known results. Some competitive algorithms[9], [11], [13], [26], [27] are compared on this platform.

5.3 Performance measures

Performance measures are crucial for quantifying the quality of Pareto Front produced by algorithms for multi-objective optimization. Convergence and diversity are the two main categories of measures. Literature [28], [29] has provided an overview of current quality measures along with their advantages and disadvantages, we select *HyperVolume(HV)*, *Maximum Spread(MS)*, *Euclidean Distance (ED)*, *Pareto Front Size(PFS)*.

Reference points are used to calculate the normal objectives and required by some quality indicators. The perfect point is the best possible point. Generally, it is an ideal point that cannot be achieved in practice. In MS-RCPSP, the duration of this point is given by summing all

durations of the tasks and then dividing this value by the number of resources, while the cost is created by assuming that each task is performed by the cheapest resource while disregarding skill constraints. Nadir Point is the point of the worst situation of each objective. The duration of this point assumes that all tasks are executed in sequence and the cost is calculated by assigning the most expensive resource to each task disregarding the skill constraints. These two points are used as the lower and upper bound to normalize the objectives.

Maximum Spread (MS) measures the spread and is given by the distance between the endpoints – a maximum extent in every dimension.
Euclidean Distance (ED) measures the average Euclidean distance between the points on obtained Pareto Front and the perfect point. It indicates the convergence of a front and a lower value implies that the front is closer to the ideal point. Note that *ED* may penalize spread because the points on the edge of the front would have a negative impact for this measure.
Pareto Front Size (PFS) measures the number of unique solutions on an obtained Pareto Front and a higher value of *PFS* is desired.

5.4 Results and comparison

To validate the effectiveness of the resource-leveling strategy, a comparison is conducted with NTGA[9] – a current known competitive algorithm for solving MS-RCPSP. For a fair comparison, each instance is run 30 times independently and results are averaged. For RLFOA, its parameter combination is selected as *NS*=10, *S*=5 and 500 generations. For NTGA, the parameter combination is given as *population_size* = 50, *generations_number* = 500, $P_m = 0.05$, $P_x = 1$, *selection_type* = tournament 2. Thus, a maximum of 2,5000 evaluation is used as the stopping criterion. All experiments are run on a PC in Windows 10 with 2.81 GHz/12 GB RAM.

Fig. 8 gives an example how Pareto Front obtained by RLFOA evolves. It can be seen that potential points generate from the right bottom corner and then extends to the left top at the beginning stage(1st – 50th generations). Then this obtained Pareto Front moves left to converge. It is worth noting that the range of Pareto Front has been largely stretched, and obviously, the number of non-dominated solutions is larger.

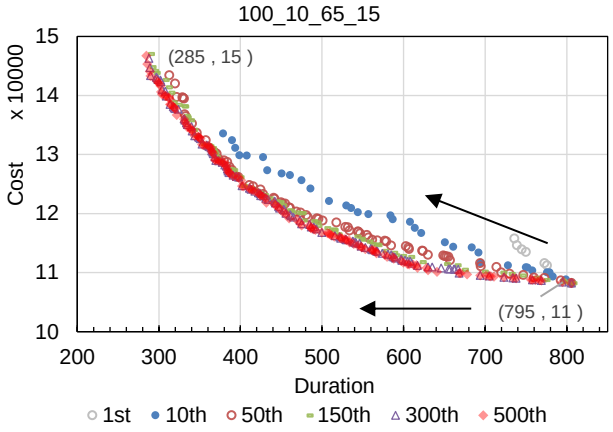


Fig. 8 Experiment results

Data presented in Table 1 shows the results of all instances by these two algorithms. Overall, RLFOA achieves better results for *PFS* in34 cases. The number of the non-dominated solutions gained by RLFOA is average 20 more than that by NTGA. *MS* has been higher for RLFOA in all cases. These two measures imply that RLFOA could achieve more non-dominated solutions with a wide expansion.

As for *ED*, NTGA has a better performance with a lower value in 26 cases. As mentioned previously, *ED* merely evaluated average distance from obtained Pareto Front to perfect point while partially disregarding the distribution of points. Fig. 9 shows a comparison of obtained Pareto Front gained by NTGA and RLFOA in instance 100_5_20_9_D3, where the difference of *ED* is the largest of these algorithms. We can see Front gained by RLFOA is far wider than that of NTGA. Points gained by RLFOA has a larger *ED* because of the stretched Front. This case also happens in other instances, hence RLFOA seems "not very well" in terms of *ED*. However, it is admitted that solutions obtained by NTGA dominates that by RLFOA at the middle of Front.

Table 1 Summary of archived results of all instances, for NTGA, RLFOA

Instance	NTGA			RLFOA		
	MS	*ED*	*PFS*	*MS*	*ED*	*PFS*
100_10_26_15	0.15702	0.36188	68.4	0.22739	0.35774	107.7
100_10_27_9_D2	0.12574	0.24088	71.23333	0.38018	0.31187	111.8
100_10_47_9	0.13599	0.31978	73.7	0.35638	0.34713	131.4333
100_10_48_15	0.11757	0.2772	60.36667	0.17268	0.28987	87.16667
100_10_64_9	0.09685	0.25361	74.76667	0.29553	0.28797	120.5
100_10_65_15	0.11907	0.35729	83.43333	0.21764	0.36509	103
100_20_22_15	0.11684	0.18312	44	0.22823	0.21403	78.26667
100_20_23_9_D1	0.09242	0.16565	36.1	0.36182	0.25434	72.1
100_20_46_15	0.12284	0.30323	40.33333	0.21668	0.30329	72.86667
100_20_47_9	0.12474	0.20103	47.6	0.37722	0.28066	100.2333
100_20_65_15	0.06002	0.20081	43.16667	0.13104	0.2208	41.86667
100_20_65_9	0.10745	0.28886	50.1	0.32066	0.31912	92.4
100_5_20_9_D3	0.19548	0.27306	183.9	0.47892	0.42125	240.9667
100_5_22_15	0.07594	0.26991	477.8	0.08597	0.2747	88.63333
100_5_46_15	0.15566	0.2949	1138.133	0.17422	0.28324	173.4333
100_5_48_9	0.12504	0.27853	218.8333	0.22002	0.31044	198.9333
100_5_64_15	0.16588	0.31698	191.9333	0.27458	0.34357	179.1667
100_5_64_9	0.1227	0.38792	114.0667	0.37772	0.44921	178.1333
200_10_128_15	0.044	0.25649	59.73333	0.14341	0.26681	122.6333
200_10_135_9_D6	0.07862	0.30318	54.46667	0.36899	0.41377	108.1667
200_10_50_15	0.08924	0.17809	49.06667	0.16058	0.20331	124.1333
200_10_50_9	0.13728	0.33475	76.46667	0.42151	0.37498	205.3
200_10_84_9	0.08949	0.22889	61.03333	0.40552	0.3038	171.4333
200_10_85_15	0.10593	0.51203	79.86667	0.25931	0.46652	175.0333
200_20_145_15	0.1015	0.31302	49.6	0.22304	0.28099	134.3667
200_20_150_9_D5	0.06574	0.2658	36.3	0.42885	0.4526	81.93333
200_20_54_15	0.07715	0.31031	50.76667	0.18747	0.28594	132.2667
200_20_55_9	0.10754	0.24606	49.13333	0.49254	0.32494	152.3333
200_20_97_15	0.07992	0.30796	39.43333	0.20505	0.29819	115.5333
200_20_97_9	0.12946	0.34602	47.9	0.33977	0.29418	149.4
200_40_130_9_D4	0.07709	0.19814	30.93333	0.31734	0.31531	87.7
200_40_133_15	0.12499	0.22467	34.16667	0.28209	0.20168	112.8
200_40_45_15	0.10033	0.2652	35.66667	0.31388	0.26208	131.3333
200_40_45_9	0.11455	0.23924	33.2	0.47219	0.31893	139.5
200_40_90_9	0.13806	0.2621	33.03333	0.41386	0.29927	146.0667
200_40_91_15	0.09682	0.23853	34.06667	0.26436	0.23314	110.3333
Average	0.110416	0.27792	107.575	0.294351	0.3119656	127.1907

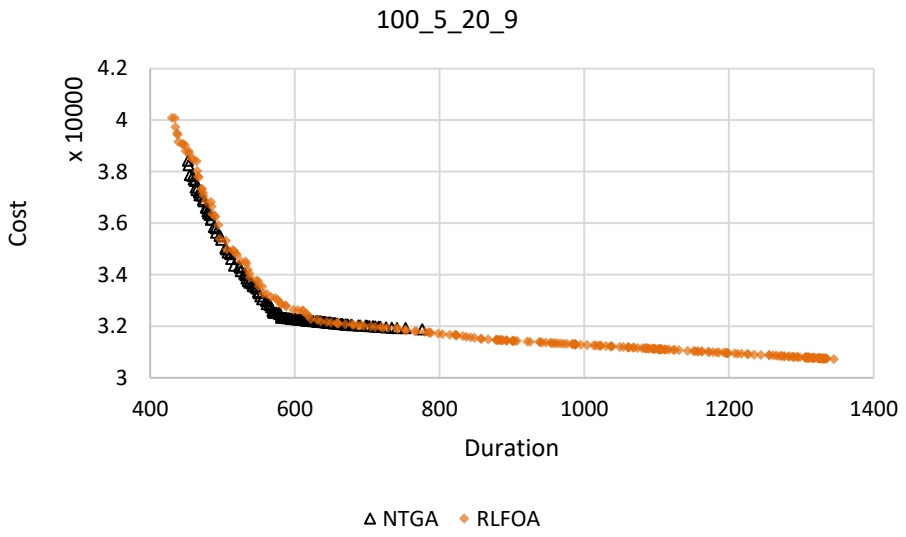


Fig. 9 Comparison of Pareto Front between NTGA and RLFOA

6 Conclusion and Future Work

In this paper, a resource-leveling strategy is proposed for solving the multi-skill resource-constrained project scheduling problem. To the best of our knowledge, this is the first time that this is the first application for MS-RCPSP. Resource load is taken into account and finding solutions can be considered as reducing the peak and fill the valley of resource utilization rate. Experiment shows the proposed strategy can generate wider Pareto Front but needs to improve in terms of convergence.

The future work is to further consider integrating the strategy to existed competitive algorithms. This strategy can significantly improve the range of Pareto Front, however, how to move this large population closer to True Pareto Front in limited generations is worth studying. Besides, it is possible to apply this strategy to other scheduling problems where certain non-dominated solutions are predefined and solutions generate by balancing the workload of resources.

Acknowledgement

The authors would like to thank Professor Pawel B. Myszkowski for providing the dataset iMOPSE and its relevant tools. This work is part of a project supported by National Key R&D Program of China (Grant No. 2018YFB1700803).

References

[1] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, “Resource-constrained project scheduling: Notation, classification, models, and methods,” *Eur. J. Oper. Res.*, vol. 112, no. 1, pp. 3–41, 1999.

[2] C. Schwindt and J. Zimmermann, *Handbook on Project Management and Scheduling Vol.1*, vol. 1. Cham: Springer International Publishing, 2015.

[3] S. Hartmann and D. Briskorn, “A survey of variants and extensions of the resource-constrained project scheduling problem,” *Eur. J. Oper. Res.*, vol. 207, no. 1, pp. 1–14, 2010.

[4] J. Węglarz, J. Józefowska, M. Mika, and G. Waligóra, “Project scheduling with finite or infinite number of activity processing modes – A survey,” *Eur. J. Oper. Res.*, vol. 208, no. 3, pp. 177–205, 2011.

[5] F. S. Al-Anzi, K. Al-Zame, and A. Allahverdi, “Weighted multi-skill resources project scheduling,” *J. Softw. Eng. Appl.*, vol. 3, no. 12, p. 1125, 2010.

[6] H. Kazemipoor, R. Tavakkoli-Moghaddam, P. Shahnazari-Shahrezaei, and A. Azaron, “A differential evolution algorithm to solve multi-skilled project portfolio scheduling problems,” *Int. J. Adv. Manuf. Technol.*, vol. 64, no. 5–8, pp. 1099–1111, 2013.

[7] O. Bellenguez and E. Néron, *Lower Bounds for the Multi-skill Project Scheduling Problem with Hierarchical Levels of Skills*, vol. 3616 LNCS. 2004.

[8] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, “Scheduling subject to resource constraints: classification and complexity,” *Discret. Appl. Math.*, vol. 5, no. 1, pp. 11–24, 1983.

[9] M. Laszczyk and P. B. Myszkowski, “Improved selection in evolutionary multi–objective optimization of multi–skill resource–constrained project scheduling problem,” *Inf. Sci. (Ny).*, vol. 481, pp. 412–431, May 2019.

[10] P. B. Myszkowski, M. E. Skowroński, Ł. P. Olech, and K. Oślizło, “Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem,” *Soft Comput.*, vol. 19, no. 12, pp. 3599–3619, 2015.

[11] H. Zheng, L. Wang, and X. Zheng, “Teaching–learning-based optimization algorithm for multi-skill resource constrained project scheduling problem,” *Soft Comput.*, vol. 21, no. 6, pp. 1537–1548, 2017.

[12] Z. Xiaolong, W. Ling, and Z. Huanyu, “A knowledge-based fruit fly optimization algorithm for multi-skill resource-constrained project scheduling problem,” in *Proceedings of the 34th Chinese Control Conference*, 2015, pp. 2615–2620.

[13] L. Wang and X. Zheng, “A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem,” *Swarm Evol. Comput.*, no. June, pp. 1–10, 2017.

[14] P. Myszkowski, L. Maciej, N. Ivan, and M. Skowroński, “iMOPSE: a library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem,” *Soft Comput.*, vol. 23, no. 10, pp. 3397–3410, 2019.

[15] P. Myszkowski, M. Skowronski, and K. Sikora, “A new benchmark dataset for Multi-Skill Resource-Constrained Project Scheduling Problem,” 2015.

[16] P. Myszkowski and M. E. Skowroński, “Specialized genetic operators for multi-skill resource-constrained project scheduling problem,” *Mendel*, pp. 57–62, 2013.

[17] B. F. Almeida, I. Correia, and F. Saldanha-da-Gama, “Modeling frameworks for the multi-skill resource-constrained project scheduling problem: a theoretical and empirical comparison,” *Int. Trans. Oper. Res.*, vol. 26, no. 3, pp. 946–967, 2019.

[18] R. Kolisch and C. Heimerl, “An efficient metaheuristic for integrated scheduling and staffing IT projects based on a generalized minimum cost flow network,” *Nav. Res. Logist.*, vol. 59, no. 2, pp. 111–127, 2012.

[19] I. Correia and F. Saldanha-Da-Gama, “A modeling framework for project staffing and scheduling problems,” in *Handbook on Project Management and Scheduling Vol. 1*, C. Schwindt and J. Zimmermann, Eds. Cham: Springer International Publishing, 2015, pp. 547–564.

[20] R. Kolisch and S. Hartmann, “Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update,” *Eur. J. Oper. Res.*, vol. 174, pp. 23–37, 2006.

[21] R. Kolisch, “Efficient priority rules for the resource-constrained project scheduling problem,” *J. Oper. Manag.*, vol. 14, no. 3, pp. 179–192, 1996.

[22] P. Myszkowski, “Novel heuristic solutions for Multi-Skill Resource-Constrained Project Scheduling Problem,” *Comput. Sci. Inf. Syst.*, pp. 159–166, 2013.

[23] P. Myszkowski, M. Laszczyk, and D. Kalinowski, “Co-Evolutionary Algorithm solving Multi-Skill Resource-Constrained Project Scheduling Problem,” 2017, pp. 75–82.

[24] J. L. Ponz-Tienda, V. Yepes, E. Pellicer, and J. Moreno-Flores, “The resource leveling problem with multiple resources using an adaptive genetic algorithm,” *Autom. Constr.*, vol. 29, pp. 161–172, 2013.

[25] M. Abdolshah, “A review of resource-constrained project scheduling problems (RCPSP) approaches and solutions,” *Int. Trans. J. Eng. Manag. & Appl. Sci. & Technol.*, vol. 5, no. 4, pp. 253–286, 2014.

[26] S. Zabihi, M. Rashidi Kahag, H. Maghsoudlou, and B. Afshar-Nadjafi, “Multi-objective teaching-learning-based meta-heuristic algorithms to solve multi-skilled project scheduling problem,” *Comput. Ind. Eng.*, vol. 136, pp. 195–211, 2019.

[27] L. Zhu, J. Lin, and Z.-J. Wang, “A discrete oppositional multi-verse optimization algorithm for multi-skill resource constrained project scheduling problem,” *Appl. Soft Comput.*, p. 105805, 2019.

[28] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen, *A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering*. 2016.

[29] M. Laszczyk and P. Myszkowski, “Survey of quality measures for multi-objective optimization: Construction of complementary set of multi-objective quality measures,” *Swarm Evol. Comput.*, vol. 48, pp. 109–133, 2019.

MISTA 2019

Flexible Outpatient Appointment Decision Considering Walk-in Patient with Dense Arrival

Mengzhong JI • Xiaojian ZHANG • Jun PENG • Yihuai HUANG • Wei XIANG*

Abstract Since walk-in patients with dense arrival are still common in most domestic hospitals, a flexible outpatient appointment considering load balancing is proposed to reduce the patients waiting time. It involves several decision variables like the booking rules, the queuing rules and the appointment rates, and forms multiple decisions. The optimal solution for different walk-in loading is determined by a multi-objective grey target decision model. The numerical experiments are implemented and the results indicate that the waiting time in peak load can be greatly reduced and several other measurements are much better than those under the pure walk-in scenario.
Keywords Appointment decision; Grey target decision model; Walk-in patients; Load balancing; Flexible appointments;

1 Introduction

Patients has to wait long time in peak time is quite common in hospital, and it is a serious problem for hospital management. Rational appointment strategies can reduce such outpatient waiting times and has been an interesting research area in hospital scientific management[1-6]. Service load is scheduled to each period by effective appointment rules[1][7][8], the length of each period is controlled, more patients can be diagnosed, the medical system operating efficiency can be improved[9-10]. Most researches focused on pure appointment study and studied how to reduce cost created by appointment patients' no show[11-12]. Some further studied the best walk-in rate in the case of no show in appointment[13], and the scheduling optimization of priority and queuing rules considering walk-in patients[14-15]. Our work targets

Mengzhong JI
Ningbo University
E-mail: 839593595@qq.com

Xiaojian ZHANG
Ningbo University
E-mail: 1115243294@qq.com

Jun PENG
Ningbo University
E-mail: 877165782@qq.com

Yihuai HUANG
Ningbo University
E-mail: 623764597@qq.com

Wei XIANG*
Ningbo University
Communication author
E-mail: xiangwei@nbu.edu.cn

in the appointment under the case of walk-in patient with peak load and further studies how to reduce waiting time in case of peak walk-in patients".

2 Load Balancing Model

In discrete manufacturing system, the discrete arrival of production orders leads to the unbalanced load, and causes order backlog when loading is higher than production capacity in some periods. Leveling production in lean manufacturing is to allocate production based on load so as to match with production capacity[16]. Such load balance theory is adopted in this study, hospital appointment rate is determined according to the walk-in patient arrival distribution, parts of the load is converted into appointment load, and the appointment load is arranged into those service capacity excess periods. Hence, the overall load is balanced with the service capacity of the hospital as shown in figure 1-2.

Hospital capacity (M_{serve}) is defined as the ratio of the average service time μ to the total duration of the period T_p. One working day is divided into several periods. X_j is the arriving patients(loading) in period $j \in \{1,2,\dots,N\}$. Patients have to wait when X_j is higher than M_{serve}.

$$\mathrm{M_{serve}} = \mathrm{int}(\frac{\mathrm{T_p}}{\mu}). \quad (1)$$

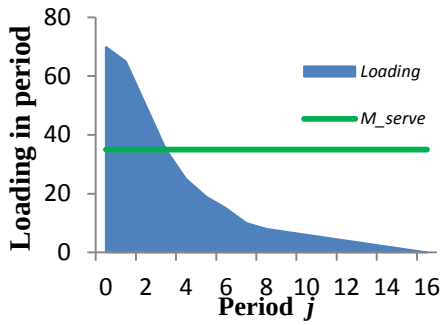


Figure 1 Real Work-load with peak

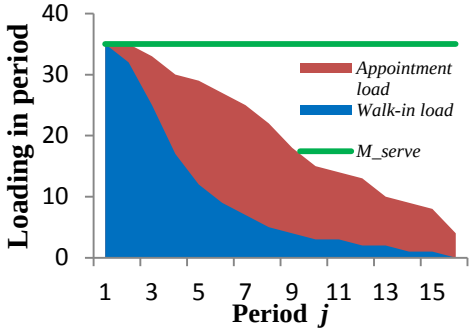


Figure 2 Work-load after leveling

A mixed approach of both walk-in and appointment patients is proposed to relieve peak pressure. Some appointment patients can be assigned to low-load periods, and most walk-in patients can get timely diagnosis and treatment. Once the appointment rate (or walk in rate (r)) is matrix, the total load can be divided into two parts: walk-in load(X_w) and appointment load(X_s). The walk-in load(X_w) has feature of dense arrival. Xw_j and Xs_j are defined as the number of walk-in patient and the number of appointment patients in period j. According to idea of load balancing and load capacity, in the jth period, if Xw_j is lower than M_{serve}, such period is considered as the excess service capacity period and can be allocated with some appointment load, however the total load should not over load capacity as shown in Eq(2) to balance total load with service capacity.

$$\mathrm{Xs_j} + \mathrm{Xw_j} \le \mathrm{M_{serve}}. \quad (2)$$

3 Appointment decision model

Appointment decision can be represented as a decision set S with A/B/C. Where, A is a set of walk-in rate, B is a set of the appointment rule and C is the queue rule set.

3.1 Set A: Walk-in Rate

The walk-in rate set $A = \{r_1, r_2, \dots, r_j\}$ stores a series of walk-in rate $r_j$at each period j. r_j can be calculated according to Eq(3-6) to ensure the walk-in patients no more than load capacity.

$$r_{j=}\begin{cases} \frac{M_{serve}}{X_j} & X_j > M_{serve} \\ 1 & X_j \le M_{serve} \end{cases}. \quad (3)$$

$$Xw_j = int(X_j * r_j). \quad (4)$$

$$Xs_j = max\{0, M_{serve} - Xw_j - max\{0, Xw_{j-1} - M_{serve}\}\}. \quad (5)$$

$$s.t\begin{cases} Xs_{sum} \ge \sum_{j=1}^{TT/Tp} Xs_j \\ X_{sum} \ge \sum_{j=1}^{TT/Tp} (Xw_j + Xs_j) \end{cases}. \quad (6)$$

Hence, the total number of walk-in patients in one day and appointment patients can be calculated respectively as Eq(7) and Eq(8). The working time will then be divided into several period according to X_{sum} with each patient occupying a period.

$$Xw_{sum} = \sum_{j=1}^{TT/Tp} Xw_j. \quad (7)$$

$$Xs_{sum} = X_{sum} - Xw_{sum}. \quad (8)$$

3.2 Set B: Appointment Rule

In order to utilize the doctors' service time reasonably, every period is divided into multiple intervals by the appointment rules. The appointment rule set B $= \{b_1, b_2, \dots, b_{10}\}$ gathers 10 different appointment rules, where appointment rules $b_1 \sim b_9$ is distinguished in interval and interval capacity as shown in the table 1.Interval is the small blocks within one appointment period, while interval capacity defines the number of patients in one interval. Appointment rules can be classified into FI/OFFSET/DOME based on the length of intervals and IB/2BEG (2B) /MB based on the interval capacity[9].

Table1 $b_1 \sim b_9$ Appointment rule

Different in intervals \ Different in interval capacity	IB	2BEG	MB
OFFSET	b_1 OFFSET: 1 1 1 · · · 1 1 1 1 1	b_2 2BOGS: 2 1 · · · 1 1 1 1 1	b_3 MOBS: m=3 · · · m m
FI	b_4 IBFI: 1 1 1 · · · 1 1 1 1 1	b_5 2BEG: 2 1 · · · 1 1 1 1 1	b_6 MBFI: m=3 · · · m m
DOME	b_7 DOME: 1 1 1 · · · 1 1 1 1 1	b_8 2BGDM: 2 1 · · · 1 1 1 1 1	b_9 MBDM: m=3 · · · m m

FI: intervals are equal in length;

OFFSET: interval increases gradually;

DOME: opposite to OFFSET, interval decreases gradually.

IB: It refers to interval capacity of 1 for all interval(i.e. only one patient can be served in an interval);

2B: It refers to the interval capacity of 2 in the first interval and 1 in the rest intervals;

MB-: It refers to the interval capacity is m(m>=2)for all blocks.

b_{10} rule is a new one and it takes varieties of environmental factors such as doctor's cost and patient's waiting cost, etc[10].

3.3 Set C: Queue Rule

Queue rule determines the sequence of patients within one period. A queuing rule set: $C = \{c_1, c_2, \dots, c_4\}$ consists of 4 queue rules, where:

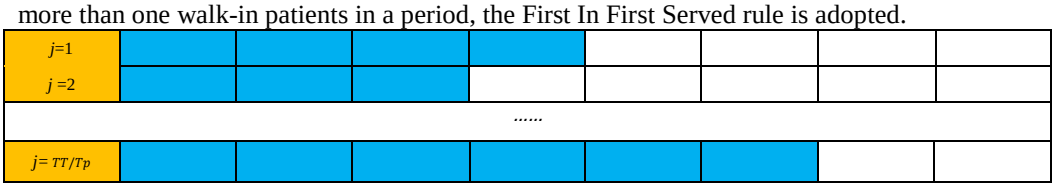
c_1 is END queue rule. It gives priority to appointment patients in one period.

c_2 is MID queue rule. It positions the appointment patients in the middle of a period.

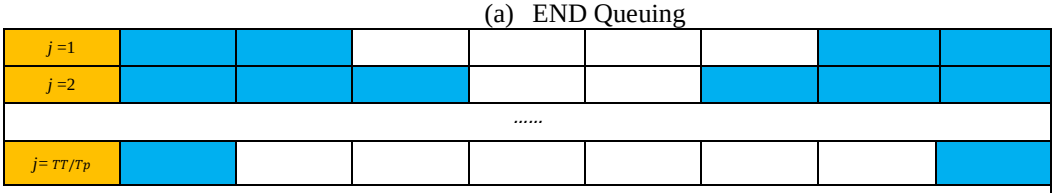
c_3 is EVEN queue rule. It allocates the walk-in patients and the appointment patients one by one.

c_4 is BEGIN queue rule. It is a new one proposed in this work, and it gives priority to walk-in patients in one period.

The detail sequence pattern in the queue under queuing rule $c_1 \sim c_4$ can be found in Figure 3. ATs_i is the *ith* appointment patient's arriving time, and ATw_{ji} is the *ith* walk-in patient's arriving time in j period. In this paper, the walk-in patient's arrival is with the negative exponential distribution. Each row represents a queue in a particular period j. Interval blocks in white are for walk-in patients, and those in blue are for appointment patients. Among these 4 rules, END, EVEN and MID can be commonly found in literatures [17]. When there are more than one walk-in patients in a period, the First In First Served rule is adopted.

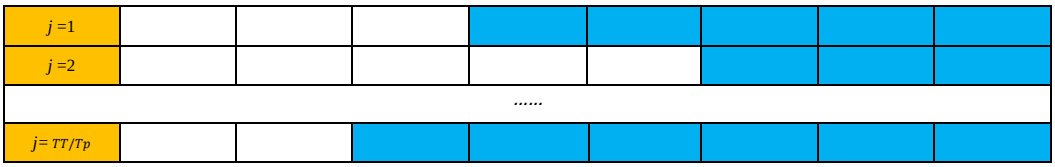


(a) END Queuing



(b) MID Queuing

(c) EVEN Queuing



(d) BEGIN Queuing

Figure 3 Queue Rule

3.4 Evaluation measurements

By selecting different decision variables in the above appointment decision model, the system will finally have different performances in term of patients' satisfactory or healthcare management. Equations (9)~(10) are defined to calculate several useful time for performance measurement. Assume *i*+1 patient service starts immediately after *i* patient's service is finished, and if *i*+1 patient does not arrive the hospital on time, the doctor remains idle.

AT_i: the arrival time of patient *i*;

ST_i: the service start time of patient *i*;

PT_i: the process time of patient *i*, *PT* is with a distribution(normal in this paper);
WT_i: the wait time of patient *i*;

$$\begin{cases} ST_1 = AT_1 \\ ST_i = max\{ST_{i-1} + PT_{i-1}, AT_i\} \end{cases}. \quad (9)$$

$$WT_i = ST_i - AT_i \quad i \in \{2,3 \dots X_{sum}\}. \quad (10)$$

Five commonly evaluation indicators are defined in this paper to evaluate each strategy [4]:
$\overline{WT}$: Average wait time;

$$\overline{WT} = \frac{\sum WT_i}{X_{SUM}}. \quad (11)$$

$\overline{WT}_w$: Walk-in patients' average wait time;

$$\overline{WT}_w = \frac{\sum(ST-ATw)}{Xw_{sum}}. \quad (12)$$

$\overline{WT}_s$: Appointment patients' average wait time;

$$\overline{WT}_s = \frac{\sum(ST-ATs)}{Xs_{sum}}. \quad (13)$$

IT: Doctor's total idle time in a working day;

$$IT = \sum_{i=2}^{X_{sum}}[ST_i - (ST_{i-1} + PT_{i-1})]. \quad (14)$$

OT: Doctor's overtime. When doctor is considered not to work overtime, it is matrix as 0;

$$OT = max\{0, ST_{X_{SUM}} + PT_{X_{SUM}} - TT\}. \quad (15)$$

4 Multi-objective Grey Target Decision Method

As mentioned in above section, the appointment decision model *S* involves three sets of the walk rate, the appointment rules and the queuing rules. Each scenario $s_{\alpha\beta\gamma}$ is composed by $a_\alpha/b_\beta/c_\gamma$, $\alpha \in [1,2,\dots,n]$, $\beta \in [1,2,\dots,m]$, $\gamma \in [1,2,\dots,e]$. Since every load scenarios need to find the optimal decision, the grey target method is introduced in this work as a decision making model.

Decision-making set *S*: a set formed by some alternative decisions, e.g. s_{133} is an alternative decision formed by a_1 / b_3 / c_3, corresponding to a decision with Walk-in Rate=a1/appointment rule = DOME/queue rule = EVEN.

$$S = \{s_{\alpha\beta\gamma} | \alpha \in [1,2,\dots,n], \beta \in [1,2,\dots,m], \gamma \in [1,2,\dots,e]\}. \quad (16)$$

The corresponding evaluation standard is matrix for each sub-matrix, so that, each scenario S_{ije} has *K* evaluation indicators ($d^k, k \in [1,2 \dots K]$). The alternative decision results should meet each evaluation index. Decision-making evaluation indexes similar with queuing theory model:

$$(d^1, d^2, d^3, d^4, d^5) = (\overline{WT}, \overline{WT}_w, \overline{WT}_s, IT, OT). \quad (17)$$

Effect matrix *U* is formed by each decision simulation result $u^k_{\alpha\beta\gamma}$, $u^k_{\alpha\beta\gamma}$ also is a result submit to evaluation index d^k, for example, u^2_{111} is a simulation result under OFFSET appointment rule/BEGIN queuing rules/walk-in rate $r_1$corresponds to evaluation index $\overline{WT}_w$.

$$U = \{u^k_{\alpha\beta\gamma} | \alpha \in [1,2,\dots,n], \beta \in [1,2,\dots,m], \gamma \in [1,2,\dots,e], k \in [1,2 \dots K]\} \quad (18)$$

Each decision measurement d^k has interval value limit(d^k_1, d^k_2):
1). Reduce the solution space;
2). Avoid the influence of extreme values on the final results.
When $u^k_{\alpha\beta\gamma}$ meet each limits, $S_{\alpha\beta\gamma}$ is a feasible alternative strategy.

$$u^k_{\alpha\beta\gamma} \in [d^k_1, d^k_2]. \quad (19)$$

$$S_{\alpha\beta\gamma}\{(u^1_{\alpha\beta\gamma}, u^2_{\alpha\beta\gamma}, \dots, u^k_{\alpha\beta\gamma}) | d^1_1 \le u^1_{\alpha\beta\gamma} \le d^1_2, d^2_1 \le u^2_{\alpha\beta\gamma} \le d^2_2, \dots, d^k_1 \le u^k_{\alpha\beta\gamma} \le d^k_2\}. \quad (20)$$

Vector r_0 is formed by optimal evaluation indexes (optimal effect vector). In this paper, the target value is considered as Eq(21), which means the system run smoothly without any idle or waiting.

$$r_0 = (r^1_0 = (\overline{WT_0}) = 0, r^2_0 = (\overline{WT_{W_0}}) = 0, \dots, r^k_0 = (OT_0) = 0). \quad (21)$$

Every feasible alternative decision will be compared to the target r_0: calculated off-target distance $|u_{\alpha\beta\gamma} - r_0|$. The one $S_{\alpha\beta\gamma}(a_\alpha, b_\beta, c_\gamma)$ with minimum off-target distance is the optimal strategy among all feasible alternative decisions.

$$min(|u_{\alpha\beta\gamma} - r_0|) = min\left(\left[\sum(u_{\alpha\beta\gamma}^k - r_0^k)^2\right]^{\frac{1}{2}}\right). \quad (22)$$

5 Model Validation and Result Analysis

A day with heavy service load is selected as a test case for model validation. The experimental data is tested in three experimental cases: 1.Pure walk-in case (the original data); 2. Pure appointment case with same load of case 1; 3.optimal decision proposed by our method;

Data fitted with experimental data in different periods: the number of patients (y) and periods (N) are in accordance with exponential distribution, and the walk-in patients' arrival time is with poisson distribution.

$$y = 0.45321e^{-\frac{N}{1.18789}} + 0.03479. \quad (23)$$

Five measurements: 1.Average wait time; 2. Appointment's wait time; 3.Walk-ins' wait time; 4.Idle time; 5. Overtime are used to compare three cases. The comparison results are analyzed in table 2 and Figure 4:

Table 2 Cases Comparison

	Walk-in	Optimal solution	Appointment
Average wait time	146.33	29.75	7.13
Appointment's wait time	0.00	12.96	7.13
Walk-ins' wait time	146.33	72.14	0.00
Idle time	1.77	11.02	10.69
Overtime	69.23	78.49	78.14

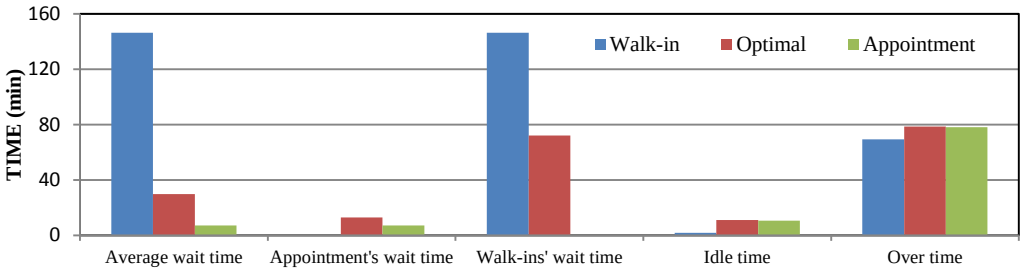


Figure 4 Comparison results in 5 measurements

1) Average wait time: The waiting time of optimal solution is 79.67% lower than that of walk-in plan. Optimal solution still has a gap to pure appointment plan in term of the waiting time, however, it has advantage of accepting the walk-in demand;

2) Waiting time for appointment patient: Optimal solution case records 12.96 min of the waiting time which is close to that of appointment case (7.13min).

3) Wait time for walk-in patient: Such measurement in walk-in plan is 146.23min and is much higher than optimal solution (72.14min), and is nearly reduced 50.66%;

4) Idle time: In walk-in case, patient receive treatment by doctor successively, server idle time is close to 0(1.77min). In optimal solution case (11.02min) and appointment case(10.69min), booking patients arrive on time, once preceding patient has a short process time, while succeeding patient had not arrived yet, doctor will be idle;

5) Overtime: In walk-in case, doctors provide service continuously until all patients are served. Overtime (69.23min) in this case is mainly determined by the number of patients and it is shorter than that of appointment case (78.14min) and optimal case (78.49min). In appointment case and optimal case, appointment patients arrive in a fixed time. However, their

service starting time may not be fixed due to the queue. Such wait time is accumulated until the end of the last patient's service and leads to longer overtime.

It can be found, the flexible appointment decision model in this paper has obvious advantages in several evaluation indicators. The idea of load balance introduced in appointment decision is proved effective. Patient load is distributed to available service periods reasonably, it reduces idle and wait time (caused by unbalance distribution of load). The resource utilization in the proposed model is more efficient, some service load in dense arrival period can be re-allocate as the appointment patients to the excessive service capacity in previous low loading period. When load(patients) wait too long, adding server(doctor) is the first choice always, in past hospital management method. However, loading balance be proved in reduce wait time usefully, it also looks like an effective way to avoid cost increase, now.

6 Conclusion

In real life, patients have to wait in a long queue for healthcare service is always a headache for healthcare service providers. Such long queue may even worse in certain peak hours with heavy load.

A mixed model of walk-in and appointment was proposed based on load-capacity balancing idea. When walk-in load exceeds the service capacity, the exceeding portion is transformed to appointment load, and the appointment load is assigned to the excess capacity periods to fully utilize the medical resources. It is different from static or dynamic scheduling. Static scheduling wastes capacity because of unbalanced load, while dynamic scheduling caused capacity unbalance, also resulting in resource waste.

Flexible appointment model is proposed based on several decision variables bank and grey target decision method. Optimal appointment decision model is suggested according to the load and distribution, and the optimized decision strategy can be obtained. The results show, optimize strategy has a better performance in five measurements. The optimize strategy has a less sensitive in resource improvement, but make full use of it.

References

1. Cayirli T, Veral E. Outpatient scheduling in health care: a review of literature, Production and Operations Management, 12 (4), 519-549(2003).
2. Ahmadi-Javid A, Jalali Z, Klassen K J. Outpatient Appointment Systems in Healthcare: A Review of Optimization Studies, European Journal of Operational Research, 258(1), 3-34(2016).
3. Gupta D, Denton B. Appointment scheduling in health care: Challenges and opportunities. IIE Transactions, 40(9), 800-819(2008).
4. Marynissen J, Demeulemeester E. Literature review on multi-appointment scheduling problems in hospitals, European Journal of Operational Research, 272(2), 407-419(2019).
5. YU Xin, LI Na, JIANG Zhibing. A Survey on the Research of Missed Appointments' Behavior and Decisions in Health System, Industrial Engineering and Management, 21(2), 132-137(2016)
6. LIANG Feng, WANG Ningbo, SHI Chenyou, et al. Analysis of Management Problems in Outpatient Appointment System, Chinese Hospital Management, 35(9), 28-31(2015).
7. Robinson L W, Chen R R. A comparison of traditional and open-access policies for appointment scheduling, Manufacturing & Service Operations Management, 12 (2), 330-346(2010).
8. Denton B, Gupta D. A Sequential Bounding Approach for Optimal Appointment Scheduling, IIE Transactions, 35(11), 1003-1016(2003).
9. Cayirli T, Veral E, Rosen H. Designing appointment scheduling systems for ambulatory care services, Health Care Management Science, 9(1), 47-58(2006).
10. Cayirli T, Yang K K, Quek S A. A Universal Appointment Rule in the Presence of No-

Shows and Walk-Ins, Production and Operations Management, 21(4), 682-697(2012).
11. Begen M A, Queyranne M. Appointment Scheduling with Discrete Random Durations, Mathematics of Operations Research, 36(2), 240-257(2011).
12. El-Sharo, Moh'd, Zheng B, Yoon S W, et al. An overbooking scheduling model for outpatient appointments in a multi-provider clinic, Operations Research for Health Care, 6(1), 1-10(2015).
13. Qu X, Peng Y, Shi J, et al. An MDP model for walk-in patient admission management in primary care clinics, International Journal of Production. Economics, 168(1), 303-320(2015).
14. Green L V, Savin S, Wang B. Managing Patient Service in a Diagnostic Medical Facility, Operations Research, 54(1), 11-25(2006).
15. Kolisch R, Sickinger S. Providing radiology health care services to stochastic demand of different customer classes, Or Spectrum, 30(2), 375-395(2008).
16. CHAO Yongsheng, SUN Wenlei, YUAN Liang. Parallel assembly line balancing based on multi-objective optimization, Computer Integrated Manufacturing Systems, 22(5), 247-250(2016).
17. Borgman N J, Vliegen I M H, Boucherie R J, et al. Appointment scheduling with unscheduled arrivals and reprioritization, Flexible Services & Manufacturing Journal, 30(1), 1-24(2017).

MISTA 2019

Single Machine Scheduling with Non-availability Interval and Optional Job Rejection

Mor Baruch · Shapira Dana*

Abstract This paper studies the single machine scheduling problem with availability constraints and optional job rejection. We consider the non-resumable and resumable variants, and show that the problems remain ordinary NP-hard, even with the rejection possibility extension, by presenting pseudo-polynomial dynamic-programming solutions. We also present an enhanced running time to the resumable scenario without job rejection.

1 Introduction

In reality, machines should undergo preventive maintenance from time to time, motivating the research on a machine setting with availability limitations. This paper studies the problem of scheduling on a single machine with availability constraints and optional job rejection. Previous research in deterministic scheduling considers two different scenarios for processing jobs on a machine with a non-availability interval:

1. *Non-resumable scenario* if a particular job is interrupted in the middle by the down period of a machine, the same job should restart right after the break.
2. *Resumable scenario* if the interrupted job can be shifted to another available machine or continue on the same machine, right after the break, from the point at which it had stopped.

In this paper we have implemented the dynamic programming (DP) solutions presented by Kellerer et al. [3] for both paradigms and designed pseudo-polynomial solutions to the same problems where it is also possible to reject some jobs, given a limit on the amount of refusal costs that can be incurred, i.e, in a case where each task has an (additional) rejection cost e_j, $1 \leq j \leq n$ and there is a total upper bound rejection

* Corresponding Author

Mor Baruch
Ariel University
E-mail: baruchm@g.ariel.ac.il

Shapira Dana
Ariel University
E-mail: shapird@g.ariel.ac.il

cost U. Our goal is to minimize the target function within the restriction that the total of rejection cost of the jobs that are not processed is less than or equal to U.

1.1 Problem Description

Given n jobs to be processed on a single machine, and a time interval $I = [s, t]$ in which the machine is disabled, the processing of each job $j \in \mathcal{J}$ takes p_j time units, and each job is assigned a weight w_j indicating its relative importance. All variables n, p_j, w_j, s, t belong to $\mathbb{N}$, and the goal is to minimize $\sum_{i=1}^{n} w_j C_j$, where C_j is the completion time of job j. The non-resumable and resumable cases are formally denoted by $1|h(1), N - res|\sum w_j C_j$ and $1|h(1), Res|\sum w_j C_j$, resp. in which $h(1)$ is used to describe a single interval.

1.2 Related Research

Adiri et al.[1] prove that minimizing the weighted completion time scheduling problem with a non available interval is NP-hard in the non-resumable case even when all weights are equal [1]. The resumable case is shown by Lee [4] to be NP hard as well, but polynomial time in case all weights are equal. Specifically, for $1|h(1), Res|\sum C_j$ the *Shortest Processing Time first* (SPT) rule is optimal. Lee [4] proves that for both resumable and non-resumable general cases, of variable size weights, there exists an optimal schedule that follows the *Weighted Shortest Processing Time first*(WSPT) order, i.e., in non-decreasing order of $\frac{p_j}{w_j}$. The ratio of a cost of a certain schedule, $Z(S)$, and an optimal one, Z(S*), might not be bounded, i.e., $\frac{Z(S)}{Z(S*}$ can be arbitrarily large but may be bounded by 3 for certain instances [5, 2]. Wang et al. [6] have suggested a 2-approximation algorithm for the resumable case that runs in quadratic time.

The solution proposed by Kellerer and Strusevich[3] for both non-resumable and resumable scenarios as well as the corresponding FPTAS solutions are based on the problem named "Symmetric Quadratic Knapsack Problem", SQKP, which they define as follows. The goal of SQKP is to:

$$\begin{aligned} \text{minimize} \quad & z = \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j \\ & + \sum_{1 \le i < j \le n} \alpha_i \beta_j (1 - x_i)(1 - x_j) \\ & + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} v_j (1 - x_j) + \Gamma \\ \text{subject to} \quad & \sum_{j=1}^{n} \alpha_j x_j \le A, \end{aligned} \tag{1}$$

where $x_j \in \{0, 1\}$, α_j, β_j, μ, v_j, and Γ are non negative integers for $j = 1, 2, \ldots, n$. The binary variables x_j can be seen as a characteristic function over the set of items, specifying the items in the knapsack. That is, x_j is equal to 1 if item j is placed into the knapsack of a total weight of at most A, while $x_j = 0$ means that the corresponding item is not placed in the knapsack.

We define a general characteristic function over all jobs indicating the accepted and rejected jobs. Formally, $\mathcal{J} = \{1, 2, \ldots, n\}$ defines the set of items, and two complementary disjoint subsets $\mathcal{A}$ and $\mathcal{R}$ of $\mathcal{J}$, i.e., $\mathcal{A} \cup \mathcal{R} = \mathcal{J}$ and $\mathcal{A} \cap \mathcal{R} = \emptyset$.

$$\chi_j = \begin{cases} 1 & \text{if } j \in \mathcal{A} \\ 0 & \text{if } j \in \mathcal{R} \end{cases} \tag{2}$$

The non-availability interval problems with optional job rejection can be rewritten as an extension of SQKP as follows.

$$\begin{aligned} \text{minimize} \quad & z = \sum_{1 \le i < j \le n} \chi_i \chi_j \alpha_i \beta_j x_i x_j \\ & + \sum_{1 \le i < j \le n} \chi_i \chi_j \alpha_i \beta_j (1 - x_i)(1 - x_j) \\ & + \sum_{j=1}^{n} \chi_j \mu_j x_j + \sum_{j=1}^{n} \chi_j v_j (1 - x_j) + \Gamma \\ \text{subject to} \quad & \sum_{j=1}^{n} (1 - \chi_j) \alpha_j x_j \le A, \end{aligned} \tag{3}$$

This generalized version of SQKP reflects the case in which there is an additional bound on the total weight of the items which are left outside the knapsack.

Kellerer and Strusevich provide a DP solution to SQKP and present a reduction from their proposed DP and subsequent FPTAS solutions to both $1|h(1), N - res|\sum w_j C_j$ and $1|h(1), Res|\sum w_j C_j$ problems. For the completeness of this paper we provide their DP after we have converted it to the non-resumable variant. For convenience, we follow the same notation.

A set of n jobs $\mathcal{J}$ should be processed without preemption on a single machine, at most one job at a time. Given a feasible schedule S, the completion time of job $j \in \mathcal{J}$ is denoted by $C_j(S)$, or C_j for short, in case the referred schedule is clear. The goal is to minimize $Z(S) = \sum_{j=1}^{n} w_j C_j$, for any feasible schedule S. The jobs are first ordered and renumbered according to the WSPT rule. Recall that the machine is not available from time s to time t, that is, $I = [s, t]$ is the non-available interval. The *crossover* job $k \in \mathbb{N}$ is the job that is interrupted at time s, while getting processed, and is restarted at time t, i.e., at time t job k is processed with the original p_k time. Under the resumable scenario, the crossover job k is interrupted at time s and resumed at time t having total processing time p_k, summing up the processing times before and after the interval I. The characteristic function x_j used in Equations 1 and 3 is reduced to the the non-resumable disabled interval, with and without job rejection by the following definition.

$$x_j = \begin{cases} 1 & \text{if } j \text{ completes before } s \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In case job j, $1 \le j \le n$ completes before the disabled interval I, that is, before s, then $C_j = \sum_{i=1}^{j} x_i p_i$, otherwise, job j completes after time t and $Cj = t + \sum_{i=1}^{j} (1 - x_j) p_j$. The variable k, $1 \le k \le n$, denotes the number of already assigned jobs, and $y_k = \sum_{j=1}^{k} p_j x_j$ is the sum of processing times of jobs assigned before the non disabled

interval. The processing time after the disabled interval is, therefore, $P_k - y_k$, where $P_k = \sum_{i=1}^{k} p_i$. The recursive function $f(k+1, y_{k+1})$ is as follows.

$$f(k+1, y_{k+1}) = \begin{cases} \min\{f(k, y_{k+1} - p_{k+1}) + w_{k+1} \cdot y_{k+1}, & \text{if } y_{k+1} \leq s \\ \quad f(k, y_{k+1}) + w_{k+1} \cdot (t + P_{k+1} - y_{k+1})\} & \\ f(k, y_{k+1}) + w_{k+1} \cdot (t + P_{k+1} - y_{k+1}) & otherwise \end{cases} \tag{5}$$

The boundary conditions are $f(0,0) = 0$, simulating the cost in case of an empty set, and $f(k, y_k) = \infty$ for $0 < k \leq n$ and $0 < y_k \leq P_n$.

The first alternative of Equation 5 represents the option that the current job, indexed $k+1$, may be done before or after the interval, while the second alternative reflects the option that the job must start after the interval. The corresponding DP algorithm runs in $O(s \cdot n)$ time. The calculation of P_k may be done by a linear preprocessing stage that stores the values in an auxiliary array.

Our paper is constructed as follows. Section 2 presents the solution to the non-resumable disabled interval with optional job rejection. Section 3 converts the solution of Kellerer and Strusevich for SQKP to the resumabe case, and suggests an improved implementation. Section 4 presents the solution for the resumable scenario with optional job rejection.

2 Non-resumable disabled interval with optional job rejection

We present a pseudo polynomial DP algorithm for solving the non-resumable non-availability interval problem with optional job rejection based on the DP presented in Equation 5. The input includes additional variables $e_1, \ldots, e_n \in \mathbb{N}$, where e_j is the cost of rejecting job j, and an additional value $U \in \mathbb{N}$ that serves as the upper bound on the total rejection cost.

The output of the algorithm is the best partition of the given jobs into three disjoint subsets: those processed before the disabled interval, those processed after, and the subset of rejected jobs, so that the target function is minimized. As claimed by Lee [4], the scheduling order of the jobs in the first two categories is in WSPT. From now on we assume that the input is arranged in this way, that is, if $i < j$, and job i and job j are in the same category then $\frac{p_i}{w_i} \leq \frac{p_j}{w_j}$ and job i is processed before job j.

At first sight it seems that a *single* additional dimension, for tracking the current rejection cost of the assigned jobs, suffices for solving the extended problem $1|h(1), N - res, \sum e_j \leq U|\sum w_j C_j$. However, another dimension is required for tracking the completion time of the last job of the already assigned jobs after the disabled interval. The requirement of this additional dimension stems from the fact that the problem $1|h(1), N - res|\sum w_j C_j$ deals with two complementing sets, in which the completion time of one subset, defines the completion time of the other subset, while here three subsets are used. More precisely, in equation 5 we use y_k to save track of the total processing times of the already assigned jobs before the disabled interval I, and $A_k = P_k - y_k$, calculated in constant time using P_k, as the total processing times of the already assigned jobs *after* I. In our case, we define A_k to follow the total processing times of the k already assigned jobs, after the disabled interval, taking into account those jobs that are rejected.

Thus, the algorithm requires two additional dimensions, one for the current rejection cost u, and another for the total completion time of the last job of the already assigned subset of jobs of $\{1, \ldots, k\}$, after the disabled interval. The variables y_k and

A_k are used to denote the completion time of the last job of the already assigned jobs $\{1,\ldots,k\}$ before/after the disabled interval in the current schedule. The variable u denotes the sum of rejection costs of the subset of $\{1,\ldots,k\}$ of jobs that are rejected in the current schedule.

The recursive formula is initialized by the boundary condition $f(0,0,0,0) = 0$, corresponding to an empty set, zero time before and after the interval and no rejection. Solving the general case, $f(k, y_k, A_k, u)$ depends on four different cases based on two conditions. The condition $y_{k+1} = y_k + p_{k+1} \leq s$ represents the option that job $k+1$ can be processed before the disabled interval, otherwise it is processed after the disabled interval. In the former case the cost is updated by $w_{k+1} \cdot y_{k+1}$, while by $w_{k+1} \cdot (t + A_{k+1})$ in the latter case, only when $A_{k+1} - p_{k+1}$ is greater than or equal to zero. The condition $u' = u + e_{k+1} \leq U$ reflects the option that job $k+1$ may be rejected, with no additional cost, otherwise, it must be processed.

1. If $y_{k+1} = y_k + p_{k+1} \leq s$ and $u' = u + e_{k+1} \leq U$,

$$f(k+1, y_{k+1}, A_{k+1}, u') = \min \Big(f(k, y_{k+1} - p_{k+1}, A_{k+1}, u') + w_{k+1} \cdot y_{k+1}, \\ f(k, y_{k+1}, A_{k+1} - p_{k+1}, u') + w_{k+1} \cdot (t + A_{k+1}), \\ f(k, y_{k+1}, A_{k+1}, u' - e_{k+1}) \Big)$$

2. If $y_{k+1} = y_k + p_{k+1} > s$ and $u' = u + e_{k+1} \leq U$,

$$f(k+1, y_{k+1}, A_{k+1}, u') = \min \Big(f(k, y_{k+1}, A_{k+1} - p_{k+1}, u') + w_{k+1} \cdot (t + A_{k+1}), \\ f(k, y_{k+1}, A_{k+1}, u' - e_{k+1}) \Big)$$

3. If $y_{k+1} = y_k + p_{k+1} \leq s$ and $u' = u + e_{k+1} > U$,

$$f(k+1, y_{k+1}, A_{k+1}, u') = \min \Big(f(k, y_{k+1} - p_{k+1}, A_{k+1}, u') + w_{k+1} \cdot y_{k+1}, \\ f(k, y_{k+1}, A_{k+1} - p_{k+1}, u') + w_{k+1} \cdot (t + A_{k+1}) \Big)$$

4. Otherwise,

$$f(k+1, y_{k+1}, A_{k+1}, u') = f(k, y_{k+1}, A_{k+1} - p_{k+1}, u) + w_{k+1} \cdot (t + A_{k+1})$$

The dynamic programming solution is straightforward, using a four dimension table, with complexity $O(n \cdot s \cdot P_n \cdot U)$, since the variable A_k is upper bounded by P_n. The partition of the jobs into the three categories can be done by backtracking with linear time complexity.

3 Resumable DP solution

The DP proposed by Kellerer and Strusevich to the resumable scenario of minimizing the total weighted completion time, $1|h(1), Res|\sum w_j C_j$, is composed of n executions of the non-resumable DP, where at the end, the best schedule is chosen. At each iteration of the solution a single job is selected as the *crossover job*, and a non-resumable schedule that minimizes the total weighted completion time of the remaining jobs is found, i.e. by running the DP for $(1|h(1), N - Res|\sum w_j C_j)$ on the subset of $n-1$ jobs, without the crossover one. The construction of a solution to the resumable case is achieved by

inserting the crossover job just after the last job that has been scheduled before the disabled interval in an optimal schedule of the non-resumable case.

More formally, let j be the crossover job, and define $\mathcal{J}' = \mathcal{J} \setminus j$ the set with the $n-1$ remaining jobs. Denote by $Z^*_{N-Res}(\mathcal{J}')$ and by $S(Z^*_{N-Res}(\mathcal{J}'))$ an optimal solution and corresponding schedule for $1|h(1), N-Res|\sum w_jC_j$ on the set $\mathcal{J}'$ of jobs calculated by means of the recursive function of Equation 5. A solution for the resumable scenario, $1|h(1), Res|\sum w_jC_j$, for the entire set of jobs $\mathcal{J}$ is reconstructed based on the optimal solution $Z^*_{N-Res}(\mathcal{J}')$ as follows. Assume that the characteristic function x_j is the partition of jobs according to the optimal schedule $S(Z^*_{N-Res}(\mathcal{J}'))$ into those that are scheduled before the disabled interval, $I = [s,t]$, and those after, as defined in Equation 4. Referring to the subset $\mathcal{J}'$ of $\mathcal{J}$, define $y_{\mathcal{J}'}$ as the total processing time of the jobs processed before s, i.e., $y_{\mathcal{J}'} = \sum_{j' \in \mathcal{J}'} p_{j'}x_{j'}$, and define $W_{\mathcal{J}'}$ as the total weight of the jobs processed after I, i.e., $W_{\mathcal{J}'} = \sum_{j' \in \mathcal{J}'} w_{j'}(1-x_{j'})$.

The variables p and w are used to denote the processing time and weight of the crossover job j, respectively. The fraction of the processed time before the disabled interval is denoted by x, $0 \le x \le 1$, so that $p \cdot x$ is the total time processed before the interval. That is, if $p < s - y_{\mathcal{J}'}$ then $x = 1$, otherwise, $px = s - y_{\mathcal{J}'}$. The optimal value Z^*_{Res} for the resumable scenario on the entire group $\mathcal{J}$ is then computed by finding the minimum between all n alternatives of the crossover job as follows.

$$Z^*_{Res}(\mathcal{J}) = min_{j \in \mathcal{J}} \begin{cases} w(y_{\mathcal{J}'} + p) & x = 1 \\ W_{\mathcal{J}'}p(1-x) + w(t + p(1-x)) & otherwise \end{cases}. \qquad (6)$$

Enhancing Complexity by an Improved Implementation

Kellerer and Strusevich [3] assume an upper bound Z^{UB} on the optimal value for the non-resumable case, which can be computed, for example, by the FPTAS of Wang et al.[6]. The authors propose an $O(n^2 \cdot s \cdot Z^{UB})$ running-time algorithm by ignoring all values that exceed Z^{UB} ([3] page 789). We present here the recursive function $f(k+1, y_{k+1}, W_{k+1})$ for the resumable scenario that by our implementation improves the running time to $O(n^2 \cdot s \cdot W)$, where W is used to denote the total weight of the entire set of jobs, i.e. $W = \sum_{i=1}^{n} w_j$,

1. If $y_{k+1} = y_k + p_{k+1} \le s$ and $W_{k+1} = W_k + w_{k+1} \le W$

$$f(k+1, y_{k+1}, W_{k+1}) = \min \Big(f(k, y_{k+1} - p_{k+1}, W_{k+1}) + w_{k+1} \cdot y_{k+1}, \\ f(k, y_{k+1}, W_{k+1} - w_{k+1}) + w_{k+1} \cdot (t + P_{k+1} - y_{k+1}) \Big)$$

2. If $y_{k+1} = y_k + p_{k+1} > s$ and $W_{k+1} = W_k + w_{k+1} \le W$

$$f(k+1, y_{k+1}, W_{k+1}) = f(k, y_{k+1}, W_{k+1} - w_{k+1}) + w_{k+1} \cdot (t + P_{k+1} - y_k)$$

3. Otherwise,

$$f(k+1, y_{k+1}, W_{k+1}) = f(k, y_{k+1} - p_{k+1}, W_{k+1}) + w_{k+1} \cdot y_{k+1}$$

It is important to stress that Kellerer and Strusevich [3] require a bigger range of the third component in the recursive function $f(k, y_k, W_k)$, and instead of W_k, refer to $W_k \cdot p$, which is bounded by Z^{UB}. However, this upper bound can be reduced by

ignoring the value p, and referring to all possible values in the range of $[0, W]$. Our implementation does not require a precomputed upper bound Z^{UB} on the cost, and uses the fact that $W_k = \sum_{j=1}^{k} w_j \cdot (1 - x_j) \leq \sum_{j=1}^{k} w_j \leq W \leq Z^{UB}$ as Z^{UB} is sum of all weights w_j, each multiplied by the corresponding completion time, that is not less than 1. As a result, the time complexity of our implementation is $O(n \cdot s \cdot W) \cdot O(n)$, as it is applied for each job that serves as a crossover job, for a total of $O(n^2 \cdot s \cdot W)$, instead of the $O(n^2 \cdot s \cdot Z^{UB})$ running time in Kellerer and Strusevich's solution.

4 Resumable Scenario with optional job rejection

In this section we provide a pseudo polynomial DP solution for the resumable version with optional job rejection, given an upper bound U on the rejection costs e_j, $1 \leq j \leq n$.

The recursive formula is initialized by $f(j', 0, 0, 0, 0, 0) = 0$, for each $j' \in \mathcal{J}$, which is considered as the crossover job. The requirement of an additional variable A_k follows the same reasoning as for the non-resumable case. Solving the general case, $f(j', k, y_k, A_k, W_k, u)$ depends on three conditions, two of them, $y_{k+1} \leq s$ and $u' \leq U$, as in the non-resumable scenario with optional job rejection. The additional variable $W_{k+1} = W_k + w_{k+1}$, ranging over values from 0 to $W = \sum_{j \in \mathcal{J}} w_j$, as in the enhanced solution for $1|h(1), Res|\sum w_j C_j$ presented in the previous section.

As in the resumable case, $1|h(1), Res|\sum w_j C_j$, the solution when extending the problem to include optional job rejection is achieved by n executions of the non-resumable DP with optional job rejection, where at the end, the best schedule is chosen. At each iteration of the solution a single job j' is selected as the *crossover job*, and a non-resumable schedule that minimizes the total weighted completion time of the remaining jobs with optional job rejection is found, i.e. by running the DP for $1|h(1), Res, \sum e_j \leq U|\sum w_j C_j$ on the subset of $n - 1$ jobs, without the crossover one. Note that the crossover job cannot be rejected, thus at least a single job must be processed. This requirement is not a burden on the proposed solution, as the case in which all jobs may be rejected, i.e. in case $\sum_{i=1}^{n} e_j \leq U$ is trivial and is therefore ignored.

1. If $y_{k+1} = y_k + p_{k+1} \leq s$ and $W_{k+1} = W_k + w_{k+1} \leq W$ and $u' = u + e_{k+1} \leq U$,

$$
\begin{aligned}
&f(j', k+1, y_{k+1}, A_{k+1}, W_{k+1}, u') = \\
&\quad \min \Big(f(j', k, y_{k+1} - p_{k+1}, A_{k+1}, W_{k+1}, u') + w_{k+1} \cdot y_{k+1}, \\
&\qquad f(j', k, y_{k+1}, A_{k+1} - p_{k+1}, W_{k+1} - w_{k+1}, u') + w_{k+1} \cdot (t + A_{k+1}), \\
&\qquad f(j', k, y_{k+1}, A_{k+1}, W_{k+1}, u' - e_{k+1}) \Big)
\end{aligned}
$$

2. If $y_{k+1} = y_k + p_{k+1} > s$ and $W_{k+1} = W_k + w_{k+1} \leq W$ and $u' = u + e_{k+1} \leq U$,

$$
\begin{aligned}
&f(j', k+1, y_{k+1}, A_{k+1}, W_{k+1}, u') = \\
&\quad \min \Big(f(j', k, y_{k+1}, A_{k+1} - p_{k+1}, W_{k+1} - w_{k+1}, u') + w_{k+1} \cdot (t + A_{k+1}), \\
&\qquad f(j', k, y_{k+1}, A_{k+1}, W_{k+1}, u' - e_{k+1}) \Big)
\end{aligned}
$$

3. If $y_{k+1} = y_k + p_{k+1} \leq s$ and $W_{k+1} = W_k + w_{k+1} \leq W$ and $u' = u + e_{k+1} > U$,

$$
\begin{aligned}
&f(j', k+1, y_{k+1}, A_{k+1}, W_{k+1}, u') = \\
&\quad \min \Big(f(j', k, y_{k+1} - p_{k+1}, A_{k+1}, W_{k+1}, u') + w_{k+1} \cdot y_{k+1}, \\
&\qquad f(j', k, y_{k+1}, A_{k+1} - p_{k+1}, W_{k+1} - w_{k+1}, u') + w_{k+1} \cdot (t + A_{k+1}) \Big)
\end{aligned}
$$

4. Otherwise,

$$f(j', k+1, y_{k+1}, A_{k+1}, W_{k+1}, u') = f(j', k, y_{k+1}, A_{k+1} - p_{k+1}, W_{k+1} - w_{k+1}, u) + (t + A_{k+1}) \cdot w_{k+1}$$

Referring to condition (1), The first minimum argument represents the option that job $k+1$ is scheduled before s, the second argument refers to the case that job $k+1$ is scheduled after t, and the last argument reflects the case that job $k+1$ is rejected. The other conditions refer to cases in which not all options are possible, therefore referring only to the relevant alternatives. The composition of the n solutions of $f(j', n, y_n, A_n, W_n, U)$ for each job $j' \in \mathcal{J}$ is constructed to obtain a solution on the entire set $\mathcal{J}$ as in Equation 6. The complexity of the provided algorithm is $O(n \cdot s \cdot P_n \cdot W \cdot U)$ for each job $j' \in \mathcal{J}$ that serves as the crossover job, for a total of $O(n^2 \cdot s \cdot P_n \cdot W \cdot U)$ time complexity.

5 Conclusion

This paper studies the problem of scheduling n jobs on a single machine with a disabled interval and optional job rejection. We propose DP solutions to the non-resumable scenario $1|h(1), N - res, \sum e_j \leq U|\sum w_j C_j$ as well as to the resumable scenario $1|h(1), Res, \sum e_j \leq U|\sum w_j C_j$. As an additional contribution we were able to enhance the solution of Kellerer and Strusevich [3] to the resumable scenario without job rejection, from $O(n^2 \cdot s \cdot Z^{UB})$ to $O(n^2 \cdot s \cdot W)$ complexity using a different application. All DPs were implemented for verification of the proposed solutions.

References

1. Igal Adiri, John L. Bruno, Esther Frostig, and A. H. G. Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Inf.*, 26(7):679–696, 1989.
2. Imed Kacem and Chengbin Chu. Worst-case analysis of the WSPT and MWSPT rules for single machine scheduling with one planned setup period. *European Journal of Operational Research*, 187(3):1080–1089, 2008.
3. Hans Kellerer and Vitaly A. Strusevich. Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica*, 57(4):769–795, 2010.
4. Chung-Yee Lee. Machine scheduling with an availability constraint. *J. Global Optimization*, 9(3-4):395–416, 1996.
5. Chung-Yee Lee. Machine scheduling with availability constraints. In *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. 2004.
6. Sun H. Chu C Wang, G. Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133:183–192, 2005.

MISTA 2019

The General Task-based Shift Generation Problem: Formulation and Benchmarks

Nico Kyngäs · Kimmo Nurmi · Dries Goossens

Abstract In workforce scheduling, shift generation is the process of determining the shift structure, along with the tasks to be carried out in particular shifts and the competences required for different shifts. Application areas of staff rostering and shift generation include hospitals, retail stores, call centers, cleaning, home care, guarding, manufacturing and delivery of goods. The General Task-based Shift Generation Problem (GTSGP) is to create anonymous shifts and assign tasks to these shifts so that employees can be assigned to the shifts. The targeted tasks must be completed within given time windows. Tasks may have shift-local precedence constraints and transition times between tasks are considered. The goal is to maximize the total number of shifts the employees are able to execute. This paper presents a mathematical formulation and some benchmark instances for the GTSGP. We also develop some preliminary metrics to assess the difficulty of GTSGP instances. We briefly describe the PEAST algorithm, which is used to solve the benchmark instances, and report on the computational results.

1 Introduction to workforce scheduling and shift generation

Workforce scheduling is an extremely challenging and time-consuming process that every organization that has employees working on shifts or on irregular working days must manage. These organizations and their domains include hospitals, retail stores, call centers, cleaning, home care, guarding, manufacturing and delivery of goods. Good overviews of workforce scheduling are published by Ernst et al. [27], Musliu et al. [28],

Nico Kyngäs
Satakunta University of Applied Sciences
E-mail: nico.kyngas@samk.fi

Kimmo Nurmi
Satakunta University of Applied Sciences
E-mail: cimmo.nurmi@samk.fi

Dries Goossens
Department of Business Informatics and Operations Management, Ghent University
E-mail: dries.goossens@ugent.be

Di Gaspero et al. [29] and Vanden Berghe et al. [30]. A very interesting framework for the general employee scheduling problem was proposed by Kletzander and Musliu [31].

The workforce scheduling process has five main components (see Figure 1). First, *workload prediction* is the phase of determining the workload. The workload can be based on both known and predicted events. For example, the arrivals of customers can be predicted using models based on queueing theory, simulation and statistics, while known events may be gathered from current sales contracts. The events are transformed into either tasks or staff demand, depending mostly on the duration and spatial diversity of the events.

The process continues with *shift generation* that produces shifts from the given workload. The most important target is to match the shifts to the workload as accurately as possible. The generation of shifts is based on either the number of employees required to work at given timeslots or the tasks that the shifts must cover.

The generated shifts form an input for the *staff rostering* phase, where employees are assigned to the shifts. The most important issues to be considered are employees' competences and preferences, along with working and resting times laid down by either collective labor agreements or labor contracts. Staff rostering also includes the scheduling of days-off and vacations. The staffing requirements must be carefully considered in the *resource and holiday planning* phase. Holidays, training sessions and other short-term absences as well as long-term sick-leaves and forthcoming retirements have a major impact on staff rostering.

The generated staff rosters often need to be changed. *Daily rescheduling* deals with ad hoc changes that are necessary due to sick leaves and other no-shows. The organization should have an appropriate mechanism to choose suitable substitutes considering the qualifications, employment contracts, legal limitations and salaries.

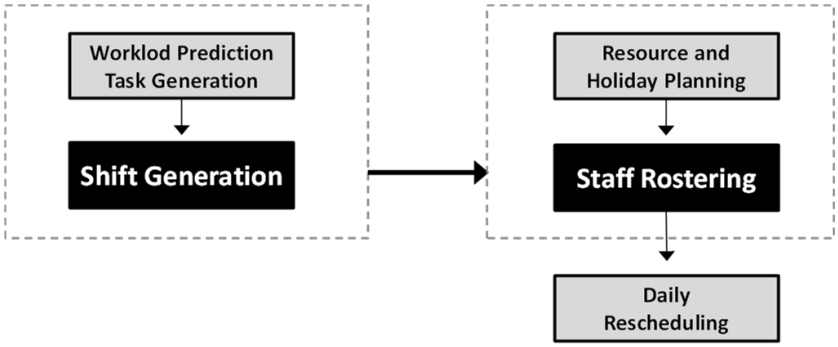


Fig. 1 The main components of the workforce scheduling process.

Academic workforce scheduling studies have mainly focused on staff rostering. In recent years, the generation of shifts has gained increasing interest. Significant benefit in financial efficiency and customer satisfaction can be achieved by optimizing the shift structure. In theory, the best results can be achieved when shift generation and staff rostering are processed and solved at the same time. However, even simplified variations of both problems and their sub-problems are known to be NP-hard and NP-complete

[18]-[22]. Nonetheless, some interesting implementations exist, see e.g. [23]-[26]. Due to the complexity issues in large-scale practical applications, shift generation and staff rostering are mainly solved separately. Our approach is to first generate the shifts and then roster the staff as described in Section 2.

The generation of shifts is based on either the varying number of required employees working during the planning horizon or the tasks that the shifts must cover. We call these employee-based and task-based shift generation problems. The first major contribution for the *employee-based shift generation* problem was the study by Musliu et al. [28]. They introduced a problem, in which the workforce requirements for a certain period of time were given, along with constraints about the possible start times and the length of shifts, and an upper limit for the average number of duties per week per employee. Di Gaspero et al. [32] proposed an employee-based problem in which the most important issue was to minimize the number of different kinds of shifts used. The problem statement also includes a collection of acceptable shift types, each of them characterized by the earliest/latest start time and minimum/maximum length of its shifts.

Bhulai et al. [33] presented a generalized model for multi-skill shift design in call centers. Their method generated a rough match between the predicted workload and labor capacity, taking the stochastic nature of the call arrival process into account. Kyngäs et al. [34] introduced the unlimited shift generation problem in which the most important goal is to minimize understaffing and overstaffing. They define a strict version of the problem, in the sense that each timeslot should be exactly covered by the correct number of employees. In [31]-[33], the shifts are limited to a number of types, for which the lengths and the start times of the shifts have to be within certain ranges. In [34], the lengths and the start times of the shifts are not strictly limited. In the person-based multitask shift generation problem with breaks presented in [35], employees can have their personal shift length constraints and competences. Even if the goal is to construct a set of shifts and not to assign them to employees, they ensure that the employees have the ability to execute the shifts.

Compared to the employee-based shift generation problem, far fewer models and algorithms have been developed for the *task-based shift generation* problem in which a number of different tasks must be carried out. The problem is to create shifts and assign tasks to these shifts so that employees can be assigned to the shifts. The first major contribution of the task-based problem was the study by Dowling et al. [24]. They created a master roster, a collection of working shifts and off shifts, and then allocated a set of tasks to personnel with the requisite skills who are available for work on that day. Krishnamoorthy and Ernst introduced a group of problems called Personnel Task Scheduling Problems (PTSP) in [36]. Given the staff that are rostered on a particular day, the PTSP is to allocate each individual task, with specified start and end time, to available staff who have skills to perform the task. Later, Krishnamoorthy et al. [37] introduced a special case referred to as Shift Minimization Personnel Task Scheduling Problem (SMPTSP) in which the only cost incurred is due to the number of personnel (shifts) that are used. This variant was motivated by situations where a large pool of casual staff is available and management would like to minimize pool usage. A similar model was earlier presented in [38] where the authors minimized the number of workers required to perform a machine load plan.

Lin and Ying [39] developed a three-phase heuristic for the SMPTSP. They obtain an initial solution using a simple but very effective construction heuristic, which is then improved using an iterated greedy heuristic, which in turn is used as an initial upper bound while solving the MIP model of the problem. Lapegue et al. [40] introduced an equity objective to the SMPTSP. The idea is to find a solution where employees have approximately the same amount of work, thus generating more fair schedules.

The next section defines the General Task-based Shift Generation Problem (GTSGP) in which the tasks are not fixed in time. Furthermore, the tasks are not explicitly assigned to employees. The main contributions of this paper are a mathematical formulation, a benchmark instance generator and computational results with PEAST, CPLEX and Gurobi for the GTSGP.

The article is organized as follows. In the next section we define and formulate the GTSGP. Section 3 introduces the first set of benchmark instances for the GTSGP. Section 4 describes the solution methods used to tackle the problem. In Section 5, we present our computational results.

2 Problem definition and formulation

In this section, we define and formulate the General Task-based Shift Generation Problem (GTSGP). To the best of our knowledge, the problem was first introduced in [41]. The motivation for the GTSGP derives from the demand-oriented workforce scheduling process presented in Section 1. As stated earlier, the complexity issues in large-scale practical applications are extremely challenging, which is why shift generation and staff rostering are mainly solved separately. Unchecked separate solving may lead to infeasibility problems in staff rostering, which we attempt to alleviate by generating shifts that can be completed by as many employees as possible, leading to increased flexibility in the rotering phase.

Given the tasks that should be rostered on a particular day, *the GTSGP is to create anonymous shifts and assign tasks to these shifts so that employees can be assigned to the shifts*. The targeted tasks must be completed within a given time window. For example, shelving in retail stores is often carried out in the forenoon. Some tasks are so-called back-office tasks. For example, in a contact center answering emails might require a given number of working hours per day dedicated to the activity but these tasks can be done any time of the day.

Recall, that we roster the staff after the shifts have been generated for each day. We consider the shift structure separately for each day, so there is no connection between the shifts of different days. We should strive to ensure that the resulting set of shifts can be carried out by the employees, i.e. each shift can be assigned to an employee while all shifts are assigned to someone and no employee is assigned to multiple shifts. For each day, the goal is to maximize the number of feasible (shift, employee) pairs. A pair (s, e) is considered feasible if employee e can carry out shift s. It is assumed that the full-time permanent and temporary employees cover 100% of the total workload.

2.1 An example problem

Figure 2 shows a simplified instance of the GTSGP with the following characteristics:

1. The planning period is divided into 19 timeslots.
2. The number of tasks is 18 and the number of employees is 8.
3. The availability of the employees during the planning period is denoted by x.
4. The durations of the tasks are given by the length of the corresponding rectangle.
5. Eleven of the tasks have time windows denoted by black lines.
6. Eight of the tasks have strict time windows, i.e. they must be carried out exactly at their given times.
7. The last task is a back-office task, i.e. its time window is the entire planning horizon.
8. The employees skilled to carry out a task are indicated in the rectangle.

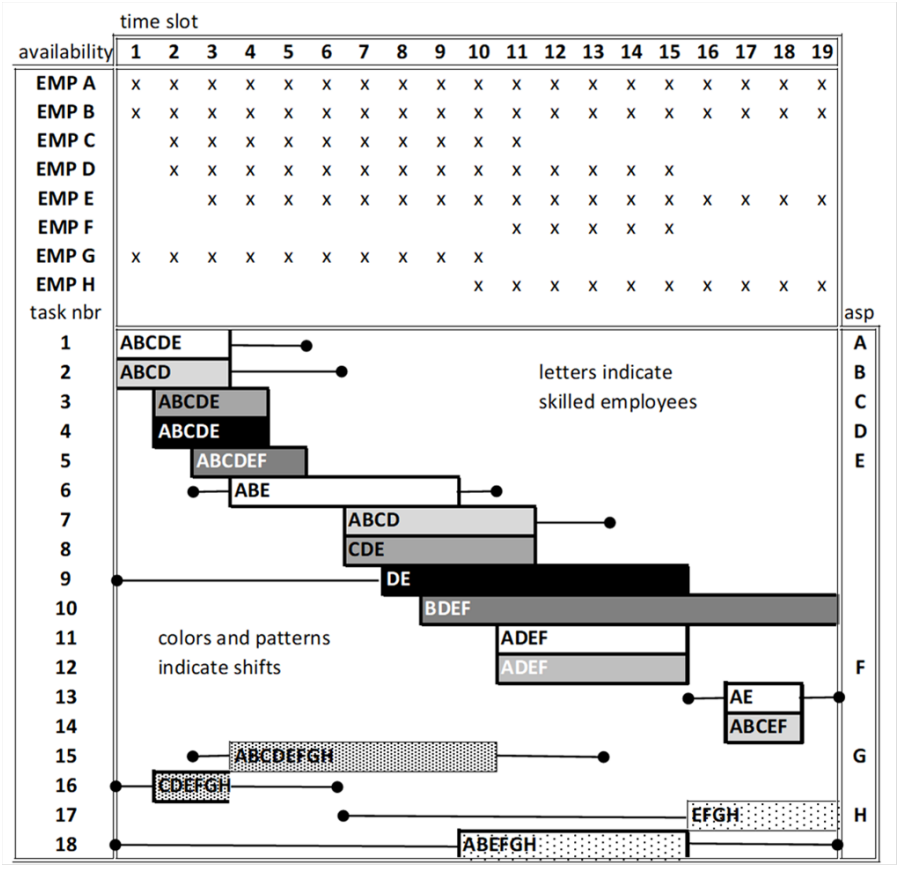


Fig. 2 A simplified instance of the GTSGP and a feasible solution to the instance. The ASP solution column shows the first task of each employee's shift.

The GTSGP differs from the SMPTSP described in Section 1 in several ways:

(G1) Tasks are not explicitly assigned to employees.

(G2) Tasks are not fixed in time.

(G3) Tasks may have shift-local precedence constraints, e.g. in case shifts of certain types are assigned to the same shift, one must be carried out before the other.

(G4) Transition times between tasks, e.g. travelling from the location of one task to the location of the next using some transportation, are considered.

(G5) The number of feasible (shift, employee) pairs is maximized.

```
Shift 1: Task #1(1)    Task #6(4)   Task #11(11)    Task #13(17)
Shift 2: Task #2(1)    Task #7(7)   Task #14(17)
Shift 3: Task #3(2)    Task #8(7)
Shift 4: Task #4(2)    Task #9(8)
Shift 5: Task #5(3)    Task #10(9)
Shift 6: Task #12(11)
Shift 7: Task #16(2)   Task #15(4)
Shift 8: Task #18(10)  Task #17(16)
```

Fig. 3 The feasible solution of the instance shown in Figure 2.

```
      Shift #1 #2 #3 #4 #5 #6 #7 #8
Employee #1  1* 1  0  0  0  1  0  0
Employee #2  0  1* 0  0  0  0  0  0
Employee #3  0  0  1* 0  0  0  0  0
Employee #4  0  0  1  1* 0  1  0  0
Employee #5  0  0  0  0  1* 1  0  0
Employee #6  0  0  0  0  0  1* 0  0
Employee #7  0  0  0  0  0  0  1* 1
Employee #8  0  0  0  0  0  0  0  1*
```

Fig. 4 The feasible pairs of the instance shown in Figure 2.

As an example, consider the instance shown in Figure 2. The colors and patterns of the tasks indicate which tasks are assigned to which shifts. A text version of the feasible solution shown in the figure is given in Figure 3. The starting times of the tasks given in parentheses correspond to the pairing between employees and shifts given in Figure 4, i.e. each shift could be carried out by at least the corresponding employee if the parenthesized task starting times were used. Starting times in the problem are actually employee-dependent and are not considered a part of the shifts.

Figure 4 shows the feasible (shift, employee) pairs of the instance. The number of feasible pairs is 14. For example, employee #1 can carry out the shifts 1, 2 and 6. Likewise, shift #3 can be carried out by employees 3 and 4. Furthermore, each shift can be assigned to an employee while all shifts are assigned to someone and no employee is assigned to multiple shifts. The asterisks denote the one and only such assignment.

2.2 Notation

S	The resulting set of shifts.
TS	The planning horizon, i.e. the contiguous set of timeslots.
T	The set of tasks.
E	The set of employees.
L	The set of locations.
C	The set of skills.
R	The set of transport types.
G	The set of task types or task groups.
M	The travel times of possible transitions. For any given transport type, the travel times must adhere to the triangle inequality, yet multiple modes of transport may be used.
P	The set of shift-local precedence constraints.
c_e	The cost of using employee e.
elb_e	Lower bound (incl.) on total working time of employee $e \in E$.
eub_e	Upper bound (incl.) on total working time of employee $e \in E$.
l_e^s	Starting location of employee $e \in E$.
l_e^e	Ending location of employee $e \in E$.
r_e	Transport type of employee $e \in E$.
a_e^s	Earliest timeslot (incl.) of availability of employee $e \in E$.
l_e^s	Latest timeslot (incl.) of availability of employee $e \in E$.
d_t	Duration of task $t \in T$.
tlb_t	Earliest timeslot (incl.) for starting time of task $t \in T$.
tub_t	Latest timeslot (incl.) for ending time of task $t \in T$.
g_t	Task type of task $t \in T$.
l_t	Location of task $t \in T$.
ttc_r	Task transition working time coefficient of transport type $r \in R$ ($ttc_r \in [0, 1]$). Transition time between two tasks is considered working time with ttc_r as coefficient, i.e. working time = transition time * ttc_r.
etc_r	Employee transition working time coefficient of transport type $r \in R$ ($etc_r \in [0, 1]$). Transition time to/from the first/last task of a shift is considered working time with etc_r as coefficient, i.e. working time = transition time * etc_r.
eta_r	Employee transition availability requirement of transport type $r \in R$ ($eta_r \in \{0, 1\}$), i.e. must the first/last transition of a shift be considered with respect to the assigned employee's availability?
$m_{rll'}$	Travel time from location $l \in L$ to location $l' \in L$ using transport type $r \in R$ ($m_{rll'} \in M$).
$p_{g,g'}$	Shift-local precedence constraint that dictates tasks of task type $g \in G$ cannot be succeeded by tasks of task type $g' \in G$.
T_e	The set of tasks doable by employee $e \in E$ ($T_e \subseteq T$).
E_t	The set of employees capable of carrying out task $t \in T$ ($E_t \subseteq E$).

2.3 Problem description

An employee e can be assigned to the shift s if and only if the following criteria hold:

(C1) He/she possesses all the skills indicated by the task types of the tasks assigned to the shift (*skill*).

(C2) The total working time of the tasks in the shift is within $[elb_e, eub_e]$ (*working time*).

(C3) All the timeslots of the tasks in the shift are within $[a_e^s, l_e^s]$ (*availability*).

(C4) He/she has enough transition time to move between the tasks in the shifts (*transition time*).

Note that these criteria implicitly define the shift structure from the skills, working times, availabilities and transition times of the employees. The GTSGP has four basic assumptions:

(B1) Every task will be assigned to a shift.

(B2) Preemption of tasks is not allowed.

(B3) Each task is processed only once without interruption.

(B4) Each employee can execute only one task at a time.

A solution is feasible, if the following hard constraints have no violations:

(H1) The tasks in the shift do not overlap in time (*overlap*).

(H2) Some tasks may have shift-local precedence constraints, that is, a task may not be executed after some other task if they are assigned to the same shift (*precedence*).

(H3) Each shift can be executed (C1-C4 hold) by one or more employees (*shift*).

(H4) Each shift can be assigned to an employee s.t. all shifts are assigned to someone and no employee is assigned to multiple shifts (*combination*).

Note that the criterion H4 actually implies H3.

The GTSGP can now be stated as follows:

1. Satisfy the hard constraints H1-H4.
2. – Maximize the sum of number of feasible (shift, employee) pairs over all pairs.
   – Minimize employee costs.
   – Minimize travelling time.

2.4 Problem formulation

In this section, we propose an integer programming formulation for the general task-based shift generation problem. In order to ensure that hard constraint (H4) is satisfied without having to solve a separate assignment problem, we opt to have a dedicated (possibly empty) shift $s(e)$ for each employee e. This enforces that, if we can feasibly assign all tasks to shifts, it will be possible to assign a shift to each employee such that all shifts are assigned to someone in the staff rostering phase (although employee e need not necessarily receive shift $s(e)$ at that point).

The following decision variables are used in the model:

$$x^s_{et} = \begin{cases} 1 & \text{if for shift } s(e) \text{ the first task is } t, \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{ett'} = \begin{cases} 1 & \text{if task } t \text{ is immediately followed by task } t' \text{ in shift } s(e) \text{ with } t \neq t' \\ 0 & \text{otherwise.} \end{cases}$$

$$x^f_{et} = \begin{cases} 1 & \text{if for shift } s(e) \text{ the last task is } t, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_{et} = \begin{cases} 1 & \text{if task } t \text{ belongs to shift } s(e) \\ 0 & \text{otherwise.} \end{cases}$$

$$sx_e = \begin{cases} 1 & \text{if shift } s(e) \text{ is non-empty} \\ 0 & \text{otherwise.} \end{cases}$$

y_{et} = starting time (i.e. first slot) of task $t \in T$ assuming the shift it belongs to is assigned to employee $e \in E$.

$wt_{ee'}$ = working time for employee $e \in E$ assuming they carry out the shift of employee $e' \in E$.

$$ve_{e'e} = \begin{cases} 1 & \text{if employee } e \text{ may not carry out shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

$$vs_{e'e} = \begin{cases} \geq 1 & \text{if employee } e \text{ does not have the skills for shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

$$vwu_{e'e} = \begin{cases} \geq 1 & \text{if shift } s(e') \text{ has too much working time for employee } e \\ 0 & \text{otherwise.} \end{cases}$$

$$vwl_{e'e} = \begin{cases} \geq 1 & \text{if shift } s(e') \text{ has too little working time for employee } e \\ 0 & \text{otherwise.} \end{cases}$$

$$vtu_{e'et} = \begin{cases} \geq 1 & \text{if employee } e \text{ would be late at task } t \text{ if carrying out shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

$$vteu_{e'et} = \begin{cases} \geq 1 & \text{if employee } e \text{ would be late at their end location due to} \\ & \text{the last task } t \text{ of shift } s(e') \text{ if carrying out shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

Problem-GTSGP

$$Z_{GTSGP} = min\Bigg(\alpha * \sum_e sx_e c_e \tag{1}$$

$$- \beta * \sum_{e,e'} (1 - ve_{e'e}) \tag{2}$$

$$+ \gamma * \left(\sum_e wt_{ee} - \sum_t d_t \right) \Bigg) \tag{3}$$

$$\text{s.t.} \quad \sum_t x^s_{et} \leq 1 \ \ \forall e \in E \tag{4}$$

$$\sum_t x^f_{et} \leq 1 \ \ \forall e \in E \tag{5}$$

$$\sum_e b_{et} = 1 \ \ \forall t \in T \tag{6}$$

$$b_{et} = x^s_{et} + \sum_{t'} x_{et't} \ \ \forall e \in E, t \in T \tag{7}$$

$$b_{et} = \sum_{t'} x_{ett'} + x^f_{et} \ \ \forall e \in E, t \in T \tag{8}$$

$$sx_e \leq \sum_t b_{et} \ \ \forall e \in E \tag{9}$$

$$sx_e \geq b_{et} \ \ \forall e \in E, t \in T \tag{10}$$

$$y_{et} \geq tlb_t \ \ \forall e \in E, t \in T \tag{11}$$

$$y_{et} \geq a^s_e + eta_{r_e} m_{r_e l^s_e l_t} \ \ \forall e \in E, t \in T \tag{12}$$

$$y_{et'} + d_{t'} + m_{r_e l_{t'} l_t} \leq y_{et} + M_1(1 - \sum_{e'} x_{e't't}) \quad \forall e \in E, t, t' \in T \tag{13}$$

$$y_{et'} + d_{t'} + m_{r_e l_{t'} l_t} \leq y_{et} + (2 - b_{et} - b_{et'}) * M_1 \quad \forall e \in E, t, t' \in T : p_{g_t, g_{t'}} \in P \tag{14}$$

$$wt_{ee'} = etc_{r_e} \sum_t \left(m_{r_e l^s_e l_t} x^s_{e't} + m_{r_e l_t l^e_e} x^f_{e't} \right) + \sum_t \left(ttc_{r_e} \sum_{t'} m_{r_e l_t l_{t'}} x_{e'tt'} \right) + \sum_t d_t b_{e't} \ \ \forall e \in E, e' \in E \tag{15}$$

$$ve_{ee} = 0 \ \ \forall e \in E \tag{16}$$

$$vs_{e'e} \geq b_{e't}(1 - q_{et}) \ \ \forall t \in T, \forall e, e' \in E \tag{17}$$

$$vwu_{e'e} \geq wt_{ee'} - eub_e \ \ \forall e, e' \in E \tag{18}$$

$$vwl_{e'e} \geq elb_e - wt_{ee'} \ \ \forall e, e' \in E \tag{19}$$

$$vtu_{e'et} \geq y_{et} - tub_t + d_t - 1 + M_1 * (b_{e't} - 1) \quad \forall t \in T, \forall e, e' \in E \tag{20}$$

$$vteu_{e'et} \geq y_{et} + d_t + eta_{r_e} m_{r_e l_t l^e_e} - a^e_e - 1 - M_1 * (1 - x^f_{e't}) \quad \forall t \in T, \forall e, e' \in E \tag{21}$$

$$M_2 * ve_{e'e} \geq vs_{e'e} + vwu_{e'e} + vwl_{e'e} + \sum_t (vtu_{e'et} + vteu_{e'et}) \quad \forall e, e' \in E \tag{22}$$

$$x^s_{et} = x^f_{et} = 0 \ \ \forall e \in E, t \in T : t \notin T_e \tag{23}$$

$$x_{ett'} = 0 \ \ \forall e \in E, t, t' \in T : e \notin E_t \cap E_{t'} \tag{24}$$

$$x^s_{et}, x_{ett'}, x^f_{et} \in \{0, 1\} \ \ \forall e, e' \in E, t \in T \tag{25}$$

$$b_{et}, sx_e, ve_{e'e} \in \{0, 1\} \ \ \forall e, e' \in E, t \in T \tag{26}$$

The objective function consists of the weighted sum of minimizing the cost of used employees (1) and traveling time considered working time (3) and maximing so-called able pairs (2), i.e. the number of (shift, employee) pairs where the employee may carry out the shift.

The sequential structure of individual shifts is considered in equations (4) - (8). Each nonempty shift has a first task, a last task, and a possibly empty chain of tasks between them. A single task belongs to exactly one shift. Equations (9) and (10) ensure sx_e indicates nonemptiness of the shift corresponding to employee $e \in E$. Equation (9) is actually redundant from the viewpoint of the model, but it is included for the sake of clarity.

Equations (11) - (13) enforce lower bounds on starting times of tasks. The respected bounds are the earliest starting time of the task, the earliest availability of the employee and preceding task in the shift.

Equation (14) ensures shift-local precedence constraints are respected whenever constrained tasks are assigned to the same shift.

Equation (15) calculates working times for (shift, employee) pairs.

Equation (16) ensures that each employee may carry out their corresponding shift.

Equations (17) - (21) calculate the violations in individual constraints between all (shift, employee) pairs. Equation (22) composes these into indicator variables to signal whether individual pairs are compatible.

All parameters affecting y_{et} have integral values, hence y_{et} will always be integral.

The following constants were used in the model:

$M_1 = |TS| + \max_t d_t + \max_{r,l,l'} m_{rll'} + 1,$

$M_2 = 2M_1 \left(1 + \max\{n \in N : \sum_{i=1}^{n} d_{t_{P(i)}} \leq l_e^s - a_e^s + 1 \ \forall e \in E \text{ or } n = 0\}\right)$ where $P(i)$ is a permutation on task indices that maps tasks to ascending order by duration,

$$q_{et} = \begin{cases} 1 & \text{if employee } e \text{ has the right skills to do task } t, \\ 0 & \text{otherwise.} \end{cases}$$

To decrease model size and improve performance, especially in depot-based instances, it is possible to define an equivalence relation between employees that are identical with respect to all constraints pertaining time, i.e. availability, locations and transport type. This can be used to limit time calculations on the equivalence classes instead of all employees. We have omitted this representation for the sake of clarity of the model.

3 Benchmark instances

A number of benchmark instances have been published for workforce scheduling problems. The best-known benchmark data set for shift scheduling (or staff rostering) was generated by Curtois and Burke [1]. Nurse rostering benchmark instances have been created by Smet et al. [2]. Data sets for the first and second international nurse rostering competitions can be found in [3] and [4]. Nurmi [5] has introduced benchmark instances

Table 1 Seventeen parameters of the test instance generator for the GTSGP.

Set	Parameter	Set	Parameter
Volume	Nbr of timeslots	Shift	Avg nbr of employees fit for a shift
	Nbr of employees		Avg nbr of tasks in a shift
	Nbr of task types		Avg difference of shift durations (%)
	Nbr of skills		
	Min/max nbr of skills in task types	Emp	Avg difference of working times (%)
Task	Pct of fixed tasks	Trans	Nbr of transports
	Pct of back-office tasks		Prob of a location change between tasks in the same shift
	Avg task window deviation (%)		Max transition time required between tasks in the same shift
	Whether or not to use task type precedence constraints		

for the core staff rostering problem. Three sets of test instances for the shift minimization personnel task scheduling problem can be found in [6]. Benchmark instances for the workforce scheduling and routing problem can be found in [7]. This section presents the first benchmark instances for the general task-based shift generation problem.

Researchers quite often only solve some special artificial cases or one real-world case. The strength of randomly generated instances is the ability to produce instances with a wide variety of different properties. Still, they should be sufficiently simple to use in different test environments. The strength of practical cases is self-explanatory when the ultimate goal is to solve real-world problems and to integrate our optimization algorithms to scheduling software packages. However, real-world instances are often confidential and sometimes too complex to represent in academic papers. The test instances presented here were randomly generated allowing us to construct arbitrarily difficult cases. The data for the instances is available online [8].

We have generated 15 random test instances in a structured way to ensure similarities to practical cases. An algorithm performing well on certain artificial instances may not perform well on another set of artificial instances, which is why we created an elaborate random test instance generator for the GTSGP. The generator is guided by five parameter sets having a total number of seventeen parameters (see Table 1).

The generator creates instances where the number of shifts equals the number of employees. The basic idea is to construct an instance along with a feasible solution. We next describe the main elements of the generator.

First, a 0-1 matrix representing a solution to the corresponding assignment problem (ASP) is generated. This ASP solution is copied to the PAIRS solution, which consists of feasible (shift, employee) pairs. Next, the collection of skills related to each task type is generated. The transport types for the employees are generated so that each transport mode is used at least once. The global transition matrix M is initialized to zero.

For each shift the tasks are generated as follows. First, the number of tasks, the shift duration and the shift start and end times are generated. Then, for each task, the start and end times and the location are generated. Note that the total number of tasks is known only after the generator stops. For the location, the probability of a location change after the previous task is considered. If the location changes, the transition time

Table 2 Characteristics of the 16 test instances.

	EX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
#timeslots	19	10	10	20	30	30	15	100	48	20	20	30	64	64	32	64
#emps	8	5	10	10	10	10	15	15	15	20	20	30	30	30	45	60
%time	51.4	77.8	91.7	66.6	83.5	92.9	100	65.7	81.8	75.8	73.8	40.7	100	100	79.6	83.5
#tasks	18	18	48	29	88	72	43	47	139	113	100	98	285	223	239	425
@tasks	2.3	3.6	4.8	2.9	8.8	7.2	2.9	3.1	9.3	5.7	5	3.3	9.5	7.4	5.3	7.1
%fixed	44.4	38.9	70.8	3.4	14.8	19.4	100	29.8	53.2	52.2	28	1	23.5	100	28.5	52
%backoff	5.6	0	0	0	0	0	0	10.6	5	0	0	6.1	0.7	0	2.1	0.2
%overlap	21.3	22.1	15.6	31.6	6.8	12.3	51.7	31.3	6.7	10.5	11.8	17.4	6.6	22.3	8.2	12.5
%maybe	60.1	52.3	47.1	74.1	29.6	30.6	51.7	74.7	53.7	53	54.1	85.4	37	22.3	69.2	43.4
@margin	65.4	80.1	79.9	104.3	112.9	118	0	175.3	591.8	214.2	140	231.3	380.8	0	326.3	239.3
#task types	11	3	10	5	10	10	15	5	20	10	10	10	30	5	20	20
@restrict	0	1.4	13.7	1.5	3.2	5.1	0	3.6	17.9	7.2	8	21.2	6.9	0	10.9	16.1
#skills	13	4	30	20	10	10	30	15	20	30	30	10	30	20	30	60
%skill	21	81.4	65.1	57.1	12.4	7.5	56.6	56.3	23.3	11.7	27.6	19.7	16.7	57.8	9.9	1.4
#locations	1	13	1	1	1	19	1	1	19	15	1	1	21	16	10	1
#transports	0	2	0	0	0	2	0	0	3	10	0	0	5	5	3	0
@pseudo	2.5	1	0.8	-0.2	0	1.4	0	0.7	0.6	3.6	2.8	0.3	0.7	1.1	0.8	0.7
ˆpseudo	-1.2	0.1	0	-0.4	-0.1	-0.3	0	-0.2	-1.6	-1.3	-0.8	-0.7	-1.6	-0.5	-3.9	-3.3
%greedy	94.4	72.2	93.8	89.7	93.2	77.8	100	93.6	88.5	85	94	88.8	92.6	95.5	89.5	93.7

%time = average availability of the employees to carry out the tasks, @tasks = average number of tasks in a shift, %fixed = percentage of fixed tasks, %backoff = percentage of back-office tasks , %overlap = probability of two tasks overlapping with random start times, %maybe = percentage of the tasks that can overlap, @margin = average percentage margin of task time windows, @restrict = the total number of tasks that cannot be executed after some other task in the same shift divided by the total number of tasks, %skill = average percentage of the total number of tasks each employee is qualified for, @pseudo = estimation of the number of "average employees" needed to complete the tasks, ˆpseudo = estimation of the number of "busy employees" needed to complete the tasks, %greedy = percentage of the tasks feasibly assigned by the greedy method.

is assigned. For the non-zero transition time, the transition matrix is updated. The time window of the task is changed in case of a fixed task or a back-office task. Next, the task type is assigned. The skills required to carry out the shift is updated according to the assigned task type. Furthermore, the task types that follow each other in the shift are updated. Finally, the shortest path algorithm is applied to the transition matrix of the selected transport type to ensure the triangle inequality holds.

When the tasks have been generated for the shifts, we first check the task types that follow each other in the shifts and create the shift-local precedence constraints accordingly. Then we decide how many employees are skilled to carry out each shift. Next, we select the employees and update the PAIRS solution accordingly. Finally, for each employee we run through the generated shifts and assign working times and possessed skills using the information generated for the tasks in the shift. When the generator ends, the tasks in the shifts present a feasible solution to the generated instance. The PAIRS solution includes the set of feasible (shift, employee) pairs and the ASP solution is a subset of the PAIRS solution.

Table 2 shows the characteristics of the 16 test instances. The EX instance is the simplified instance shown in Figure 2. The rest of the instances are generator instances. The number of timeslots varies between 10 and 100, the number of employees between 5 and 60 and the number of tasks between 19 and 425. The transport types are used in seven instances and precedence constraints in eleven instances. Two instances have only fixed tasks, and two instances have back-office tasks.

Most of the characteristics are self-explanatory, but some need more insight. For example, the %*time* measure is 0.50, if employee#1 is always available to carry out 25% of the tasks, employee#2 50% and employee#3 75%. The calculation of the task overlapping probabilities (%*overlap* and %*maybe*) is very time-consuming. We need to iterate all task pairs and for each pair check for both tasks all possible time slot positions

within the time windows. The ^*pseudo* estimator is calculated as the sum of the task durations divided by the sum of the maximum working times of the employees. The @*pseudo* estimator uses average working times and considers the average transition times for the employees.

In order to compute %greedy, the percentage of the tasks that can be feasibly assigned to shifts by a greedy method, we use an iterative constructive heuristic. The basic idea is similar to the heuristic for the SMPTSP presented in [39]. However, the complexity of the GTSGP compared to the SMPTSP requires more advanced features. Furthermore, the greedy method uses a sophisticated iteration mechanism. The implementation details of the method are out of the scope of this study.

We believe that some of these characteristics can help us to understand the hardness of the instances. We return to this question in Section 5, where we present our computational results. First, the next section describes the solution methods used to tackle the problem.

4 Solution methods

We have created the PEAST algorithm [17] to solve challenging scheduling problems. There is evidence that the algorithm can successfully solve different problem domains. We have used it to schedule sports leagues (see e.g. [42] and [11]) and staff rosters [12] and to generate shifts [13] and school timetables [14]. Furthermore, we have used the algorithm to solve academic problems, such as balanced incomplete block design [10], single round robin tournaments with balanced home-away assignments and pre-assignments [10], days-off scheduling [15] and constraint minimum break problems [16].

The PEAST algorithm is a population-based metaheuristic. The heart of the algorithm is an ejection chain search, which is used to explore promising areas in the search space. Another important feature of the algorithm is the use of shuffling operators. They assist in escaping from local optima. Furthermore, simulated annealing refinement and tabu search are used to avoid staying stuck in promising search areas too long. The acronym PEAST stems from the methods used: Population, Ejection, Annealing, Shuffling and Tabu. We will next discuss these and other important features of the algorithm briefly. For detailed discussion we refer to [17]. The pseudo-code of the algorithm is given in Figure 5.

The ejection chain search extends a basic hill-climbing step to generate a sequence of moves in one step, leading from one solution candidate to another. The search operator moves an object, o_1, from its old position, p_1, to a new position, p_2, and then moves another object, o_2, from position p_2 to a new position, p_3, and so on, ending up with a sequence of moves. In the GTGSP, the object is a task and the current position is the relative position of the task in the list of tasks in the current shift. We refer to [17] for the detailed discussion of the object selection and the stopping criteria.

We improve the search by introducing a short-term *tabu list* which prevents reverse order moves in the same sequence of moves. i.e. if we move an object o from position p_1 to position p_2, we do not allow o to be moved back to position p_1 before a new sequence of moves begins. We use the *simulated annealing refinement* to decide whether to commit to a sequence of moves in the ejection chain search. This refinement is different from the standard simulated annealing. It is used in a three-fold manner [17]:

```
Set iteration limit t, population size n,
    elitism interval e, shuffling interval s and ADAGEN update interval a
Generate a random initial population of schedules S_i for 1 <= i <= n
Set best_schedule = null, iteration = 1
WHILE iteration ≤ t
    pop = 1
    WHILE pop <= n
        Apply ejection chain search to schedule S_pop to get a new schedule
        IF Cost(S_pop) < Cost(best_schedule) THEN Set best_schedule = S_pop
        pop = pop + 1
    END WHILE
    Update simulated annealing framework
    IF round ≡ 0 (mod a) THEN Update the ADAGEN framework
    IF round ≡ 0 (mod s) THEN Apply shuffling operators
    IF round ≡ 0 (mod e) THEN Replace the worst schedule with the best one
    Set iteration = iteration + 1
END WHILE
Output best_schedule
```

Fig. 5 The pseudo-code of the PEAST algorithm.

1. when choosing an object to be moved,
2. when choosing the new position of the object, and
3. when the sequence of moves is cut short.

The *shuffling operators* are used to perturb a solution into a potentially worse solution in order to escape from local optima. For example, a number of random objects are moved from their current positions to new random positions. Two shuffling operators are used in the GTGSP. The first one moves m tasks to random positions in random shifts. The value m is a random number between $m_{min} = 1$ and $m_{max} =$ the number of employees divided by 10, where m_{min} and m_{max} are constants determined by algorithm tuning. The other one selects from a random shift all the tasks that no employee can carry out and moves each of them to a random position in a random shift.

Due to the very large search space of the GTSGP, we have to consider two calculation issues. First, when generating a sequence of moves, we have to calculate the cost function many times during one ejection chain search. Furthermore, we have to calculate the cost function while rolling back the moves, often down to the starting point. Therefore we only recalculate those parts of the solution that are changed due to the single moves of the move sequence. We also have to solve an assignment problem to check the hard constraint H4, which can be solved in $O(n^3)$ time using appropriate data structures [43]. This is far too slow since we have to solve the problem in each single move in the ejection chain search. Fortunately, implementing the move sequence in such a way that we can apply the ideas presented in [44], we are able to calculate only the changes incurred to the initial assignment problem. In our implementation the calculation of changes requires $O(n^2)$ operations.

The PEAST algorithm uses a traditional penalty method, which assigns positive weights to the hard and soft constraints and sums the violation scores to get a single value to be optimized. The soft constraints are assigned fixed weights according to their significance. However, the hard constraints are assigned dynamic weights using the ADAGEN method described in [17].

The PEAST algorithm has been criticized of being nothing more than a collection of old ideas. To the best of our knowledge, at least the ejection chain search, simulated

annealing refinement and dynamic weights assigned to the hard constraints have been novel ideas. More importantly, each of the five components in the PEAST name are crucial to produce good-quality solutions. This was verified for three different problem domains in [17]. We will demonstrate this for the GTSGP in Section 5.

Most heuristic algorithms created to solve real-world cases use a fast greedy method to generate a sufficiently good initial solution. For the PEAST algorithm, we have found no evidence that a sophisticated initial solution improves results. On the contrary, random initial solutions seem to yield superior or at least as good results in all the real-world cases where we have applied the algorithm. We will demonstrate this for the GTSGP in Section 5.

5 Computational results

This section presents our computational results for the benchmark instances introduced in Section 3. We solved the instances using the PEAST algorithm described in Section 4. We used exactly the same version and the parameters of the algorithm that are in use in the real-world optimization in the workforce scheduling [12] and in the professional sports league scheduling [42]. We used no domain-specific knowledge in order to generate better solutions.

For the benchmark instances the number of shifts required was known a priori to be equal to the number of employees. Due to in part this, all our tests were done using weights $\alpha = 0, \beta = 1, \gamma = 0$ for the cost function. This choice leads to maximization of able pairs, which in general tends to maximize the number of shifts used.

The PEAST algorithm was able to find a feasible solution to all but one of the instances. Table 3 shows the results and the number of minutes required to reach the solution (best of ten runs). All test runs using the PEAST algorithm were carried out using an Acer Laptop on an Intel Core i7-8550 at 1.8GHz with 8GB RAM operating Windows 10. We suspect that better solutions for most of the instances exist. The data for the instances is available online [8].

Mathematical solvers were used for the sake of getting a baseline on problem difficulty. The aim was not comparing solvers against either PEAST or each other. The Gurobi runs were done with Gurobi 8.1 through the Java API on a 4GHz i7-4790K with 16GB of RAM using default settings. The CPLEX runs were done with Dell XPS 8930, with Intel Core i7-8700 @ 3.2 GHz. All solver runs were limited to 30 minutes. For instances 00, 04 and 05, GUROBI found provably optimal solutions.

We believe that some of the characteristics of the GTSGP instances presented in Section 3 could help us to understand the hardness of the instances. Of course, the number of tasks, the number of employees and the number of time slots have a direct influence on the hardness of the instance, since they enlarge the search space. To be more exact, the combinatorial search space explodes when the average number of tasks in a shift (@tasks) and the number of average employees needed to complete the tasks (@pseudo) increase. In our first test runs, the %time, @margin and %greedy measures seem to influence the hardness of the problem the most. However, we have not yet conducted enough experiments to determine what makes an instance easy or hard.

Table 3 The results for the sixteen benchmark instances.

	PEAST ofv	PEAST time (m)	GUR	GUR (b)	CPLEX	CPLEX (b)
EX	15	1				
00	6	1	6	6	x	x
01	53	1	51	56	49	57
02	25	1	25	33	22	36
03	15	1	-	42	-	50
04	22	85	22	22	x	x
05	146	1	146	146	146	161
06	100	3	-	163	-	176
07	110	21	-	-	x	x
08	-	-	-	-	x	x
09	60	1	-	264	-	352
10	205	171	-	-	-	778
11	371	65	-	-	x	x
12	441	30	-	-	x	x
13	585	59	-	-	x	x
14	615	202	-	-	x	x

PEAST ofv = best PEAST solution found, PEAST time (m) = time to find feasible solution by PEAST, GUR (b) = best value (bound) found by GUROBI, CPLEX (b) = best value (bound) found by CPLEX.
- = not found.
x = not run.

Table 4 The effect of individual PEAST components on solution quality.

Without	Solution value(*)	Without	Solution value(*)
Population	3	Elitism	11
Ejection chain search	11	ADAGEN	1
SA refinement	2	Random start(x)	1
Shuffling operators	17		
Tabu list	1		
PEAST	0		

(*) The solution value indicates the number of hard constraint violations. The value is the median of ten runs.
(x) Greedy initial solutions used instead of random initial solutions.

As stated in Section 4, each of the five components in the PEAST name are crucial to produce good-quality solutions. To demonstrate this for the GTSGP, we solved benchmark instance #8 without using each of these components. The results are shown in Table 4. The results are in line with our findings in other problem domains. The results also show that elitism and ADAGEN method produce better results. Furthermore, a greedy initial solution does not improve results. Instead, random initial solutions yield better results.

6 Conclusions and future steps

We presented the problem definition and the mathematical formulation of the General Task-based Shift Generation Problem (GTSGP). We justified the importance of the problem in the big picture of the workforce optimization and the real-world workforce scheduling process.

We introduced the first set of benchmark instances for the GTSGP, which has been made available online. We solved the instances to feasibility using the PEAST al-

gorithm. The computational results were encouraging. However, this was the first encounter of the instances and a solution method. There should be room for improvement and methods that are more efficient.

Our near future direction is to use the GTSGP test generator to create a larger set of benchmark instances to better understand the hardness of the problem.

References

1. T. Curtois and E. Burke: "Employee Shift Scheduling Benchmark Data Sets" [Online]. Available: http://www.schedulingbenchmarks.org/, (Last update 25-Aug-2017).
2. P. Smet, F. Salassa and G. Vanden Berghe. "Nurse Rostering Benchmark Instances" [Online]. Available: https://people.cs.kuleuven.be/ pieter.smet/nurserostering.html, (Last access 10-Aug-2019).
3. P. De Causmaecker: "The First International Nurse Rostering Competition" [Online], Available: https://www.kuleuven-kulak.be/nrpcompetition, (Last access 10-Aug-2019).
4. S. Ceschia, N. Dang, P. De Causmaecker, S. Haspeslagh and A. Schaerf: "The Second International Nurse Rostering Competition" [Online]. Available: http://mobiz.vives.be/inrc2/, (Last access 10-Aug-2019).
5. K. Nurmi: "The Core Staff Rostering Problem - Benchmark Instances" [Online]. Available: http://web.samk.fi/public/tkiy/CSP/, (Last update 22-Jul-2015).
6. T. Lapegue "Personnel Task Scheduling Problem Library" [Online], Available: https://sites.google.com/site/ptsplib/smptsp/instances, (Last access 21-Apr-2013).
7. J.A. Castillo-Salazar, D. Landa-Silva and R. Qu, "Workforce scheduling and routing problems: literature survey and computational study", Annals of Operations Research, 239(1), pp. 39–67, 2016.
8. K. Nurmi: "The General Task-based Shift Generation Problem - Benchmark Instances" [Online]. Available: http://web.samk.fi/public/tkiy/GTSGP/, (Last update 16-August-2019).
9. D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, pp. 67-82, 1997.
10. K. Nurmi, D. Goossens and J. Kyngäs, "Scheduling a Triple Round Robin Tournament with Minitournaments for the Finnish National Youth Ice Hockey League", Journal of the Operational Research Society, Vol. 65(11), pp. 1770-1779, 2014.
11. J. Kyngäs, K. Nurmi, N. Kyngäs, G. Lilley, T. Salter and D. Goossens, "Scheduling the Australian Football League", Journal of the Operational Research Society, 2017.
12. N. Kyngäs, K. Nurmi and J. Kyngäs, "Workforce Scheduling Using the PEAST algorithm", in Ao, Sio-Iong (ed.): IAENG Transactions on Engineering Technologies, Lecture Notes in Electrical Engineering Volume 275, Springer, USA, pp 359-372, 2014.
13. N. Kyngäs, K. Nurmi and J. Kyngäs, "Solving the person-based multitask shift generation problem with breaks", Proc. of the 5th International Conference On Modeling, Simulation And Applied Optimization, Hammamet, Tunis, pp. 1-8, 2013.
14. K. Nurmi and J. Kyngäs, "A Conversion Scheme for Turning a Curriculum-based Timetabling Problem into a School Timetabling Problem" in Proc of the 7th Conference on the Practice and Theory of Automated Timetabling (PATAT), Montreal, Canada, 2008.
15. K. Nurmi and J. Kyngäs, "Days-off Scheduling for a Bus Transportation Company", International Journal of Innovative Computing and Applications 3 (1), pp. 42-49, 2011.
16. K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngäs, J. Marenco, C.C. Ribeiro, F. Spieksma, S. Urrutia and R. Wolf-Yadlin, "A Framework for Scheduling Professional Sports Leagues", in Ao, Sio-Iong (Eds.). IAENG Transactions on Engineering Technologies 5, Springer, USA, pp. 14-28, 2010.
17. N. Kyngäs, K. Nurmi and J. Kyngäs, "Crucial Components of the PEAST Algorithm in Solving Real-World Scheduling Problems", Lecture Notes on Software Engineering 1(3), pp. 230-236, 2013.
18. M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, 1979.
19. J. Tien and A. Kamiyama, "On Manpower Scheduling Algorithms," in SIAM Rev. 24 (3), pp. 275–287, 1982.
20. H.C. Lau, "On the Complexity of Manpower Shift Scheduling," Computers and Operations Research 23(1), pp. 93-102, 1996.

21. D. Marx, “Graph coloring problems and their applications in scheduling,” Periodica Polytechnica Ser. El. Eng. 48, pp. 5–10, 2004.
22. P. Bruecker, R. Qu and E. Burke, “Personnel scheduling: Models and complexity”, European Journal of Operational Research 210 (3), pp. 467-473, 2011.
23. T. Lapegue, O. Bellenguez-Morineau and D. Prot, “A constraint-based approach for the Shift Design Personnel Task Scheduling Problem with Equity”, Computers and Operations Research 40 (10), pp. 2450-2465, 2013.
24. D. Dowling, M. Krishnamoorthy, H. Mackenzie and H. Sier, “Staff rostering at a large international airport”, Annals of Operations Research 72, pp. 125-147, 1997.
25. D. Prot, T. Lapgue and O. Bellenguez-Morineau, “A two-base method for the shift design and personnel task scheduling problem with equity objective”, International Journal of Production Research 53 (24), pp. 7286-7298, 2015.
26. P. Smet, A.T. Ernst and G. Vanden Berghe, “Heuristic decomposition approaches for an integrated task scheduling and personnel rostering problem”, Computers & Operations Research 76, pp. 60-72, 2016.
27. A.T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, “Staff scheduling and rostering: A review of applications, methods and models,” European Journal of Operational Research 153 (1), pp. 3-27, 2004.
28. N. Musliu, A. Schaerf and W. Slany, “Local search for shift design,” European Journal of Operational Research, 153(1), pp. 51–64, 2004.
29. L. Di Gaspero, J. Gärtner, N. Musliu, A. Schaerf, W. Schafhauser and W. Slany, “Automated Shift Design and Break Scheduling”, In: Uyar A., Ozcan E., Urquhart N. (eds) Automated Scheduling and Planning. Studies in Computational Intelligence, vol. 505, Springer, Berlin, Heidelberg, 2013.
30. J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester and L. De Boeck, “Personnel scheduling: A literature review”, European Journal of Operational Research 226 (3), pp 367-385, 2013.
31. L. Kletzander and N. Musliu, “Solving the General Employee Scheduling Problem”, Preprint submitted to Computers and Operations Research, Vienna, Austria, 2018.
32. L. Di Gaspero, J. Gärtner, G. Kortsarz, N. Musliu, A. Schaerf and W. Slany, “The minimum shift design problem,” Annals of Operations Research, 155(1), pp. 79–105, 2007.
33. S. Bhulai, G. Koole and A. Pot, ”Simple Methods for Shift Scheduling in Multiskill Call Centers”, Manu-facturing and Service Operations Management 10 (3), pp. 411-420, 2008.
34. N. Kyngäs, D. Goossens, K. Nurmi and J. Kyngäs, “Optimizing the unlimited shift generation problem”, Applications of Evolutionary Computation: EvoApplications, Springer, pp. 508-518, 2012.
35. N. Kyngäs, K. Nurmi and J. Kyngäs, “Solving the person-based multitask shift generation problem with breaks”, In Proc. of the 5th International Conference On Modeling, Simulation And Applied Optimization, Hammamet, Tunis, pp. 1-8, 2013.
36. M. Krishnamoorthy, A.T. Ernst and D. Baatar, “The personnel task scheduling problem”, Optimization Methods and Applications, pp. 343–367, 2001.
37. M. Krishnamoorthy and A.T. Ernst, “Algorithms for large scale Shift Minimisation Personnel Task Scheduling Problems”, European Journal of Operational Research, 219 (1), pp. 34-48, 2012.
38. V. Valls, A. Perez and S. Quintanilla, ”A graph colouring model for assigning a heterogenous workforce to a given schedule”, European Journal of Operations Research 90, pp. 285-302, 1996.
39. S.-W. Lin and K.-C. Ying, “Minimizing Shifts for Personnel task Scheduling Problems: A three-Phase Algorithm”, European Journal of Operational Research, 237, pp. 323-334, 2014.
40. Lapègue, T., Prot, D., Bellenguez-Morineau, O.: A constraint-based approach for the Shift Design Personnel Task Scheduling Problem with Equity. Computers and Operations Research 10(40), pp. 2450–2465, 2013.
41. N. Kyngäs, K. Nurmi and J. Kyngäs, ”Workforce Optimization: the General Task-based Shift Generation Problem”, Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2019, 13-15 March, 2019, Hong Kong, pp. 510-515, 2019.
42. K. Nurmi, J. Kyngäs and A.I. Järvelä, “Ten-year Evolution and the Experiments in Scheduling a Major Ice Hockey League”, in Daniel Hak (ed.): An in Depth Guide to Sports, Nova Science Publishers, USA, pp 169-207, 2018.
43. C.H. Papadimitriou and K. Steiglitz, “Combinatorial Optimization: Algorithms and Complexity”, Dover Publications, 1998.
44. I.H. Toroslu and G. Ucoluk, “Incremental assignment problem”, Information Sciences 177(6), pp. 1523-1529, 2007.

MISTA 2019

Two algorithms for the Flexible Job Shop Scheduling Problem with limited transportation capacity

Chengfeng Peng • Zhantao Li • Qingxin Chen • Jianjun Liu

Abstract This paper takes the intelligent manufacturing workshop with Automatic Guided Vehicle (AGV) as the research background and research the Flexible Job Shop Scheduling Problem (FJSSP) with limited transportation capacity. The optimization criterion considered is the minimization of total completion time. We decompose the problem into three phases: job selection, machine selection and AGV selection. Then we show a robust genetic algorithm (GA) and a simulating annealing algorithm (SA) base on the three phases to solve the problem. A complete evaluation of 126 operators for GA and 7 operators for SA makes an internal comparison. For the evaluations we use four different kinds of layout. The result show that the proposed algorithms are very effective and at same running time are easy to implement.

1 Introduction

With the increased for manufacturing automation and intelligence, as the important manufacturing resource, AGV has becoming more and more important because it can improve the flexibility of the manufacturing system. But higher flexibility also brings a serious challenge for shop scheduling problem, without an appropriate scheduling strategy, it can't get the high resource utilization then improve the system productivity. This is an interesting problem to research.

The FJSSP with limited transportation capacity scheduling problem is also a simultaneous scheduling problem between processing equipment and transport equipment in flexible job-shop. And the limited transportation capacity means there are some conflicts need to be considered during the shop scheduling. It is a highly complex issue worth studying.

Chengfeng Peng
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: 286071804@qq.com

Zhantao Li
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: lzt0@163.com

Qingxin Chen
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: qxchen@gdut.edu.cn

Jianjun Liu
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: jianjun33@163.com

Sabuncuoglu [1] proposed an on-line dispatching algorithm which schedules the jobs on machine or an AGV one at time as the scheduling decision is needed in FMS. Knust [2] extend the α|β|λ notation for scheduling problems and analyze the complexity of shop-scheduling problems with transportation, proved the problem J, $R_1|t_{jkl}, t'_{kl}|C_{max}$ is NP-hard then propose a TS algorithm to solve the problem. Rossi [3] proposes a swarm intelligence approach based on a disjunctives graph model to solve the job shop scheduling with transportation times, but in this paper, it seen the transportation with no-conflicts. Ulusoy [4] decomposed the problem that simultaneous scheduling of machines and AGV into two subproblem, and use the heuristic to solve it. He [5] proposed robot and machine scheduling rules with a state-dependent part input sequencing algorithm to solve the MS with AGV problem. Based on the grid layout environment, Mehraba [6] proposed a two stage Ant Colony Algorithm to solve the conflict-free routing problem for AGVs. Zheng [7] solve the simultaneous machine/AGV scheduling problem with a heuristic based on tabu search (TS) to get optimal or near to optimal solution for large-size problems within reasonable computation time.

Current researches on simultaneous scheduling problems are focus on the no-conflicts shop-scheduling problems with transportation. But to solve the shop-scheduling problem with transportation conflicts in flexible job shop we should not only consider the availability of the machines but also consider the influence of process path and layout.

2 Problem structure

Figure 1 represents a FJSSP with transportation consider the conflicts. The problem environment and assumptions of the present work are given below:

Figure 1 The FJSSP with transportation

2.1 Flexible Job Shop (FJS) environment

The environment within which the FJS under consideration can be described as follows: There are a set of jobs., and each job has its operations. For each operation of one job it can be process by a set of machines only once with different process time and the machine also can process other operation.

2.2 Assumption

(1) The types and numbers of machines and AGVs are known. All of the operations are non-preemptive. The input/output buffer space at each machine is limitless.

(2) Processing time, set-up time, loading and unloading time are available and are deterministic.

(3) Numbers of AGVs is given and AGVs are all identical in the sense that they have the same speed and load carrying characteristics.

(4) The layout of workshop is given and choose the shortest path as the move path. All travel times on each position are known.

(5) A load/unload(L/U) stating serves as a distribution center for parts not yet processed and as a collection center for parts finished. And all vehicles start from the L/U station initially and return after finishing al their assignments. The buffer space of input/output (I/O) is unlimited and the station of each I/O have given.

(6) each AGV carry a single unit-load at a time. Preemption of moves is not allowed. The trips are called loaded or empty trips depending on whether a part is carried or no part is carried during this trip, respectively. The durations for the deadheading trips are sequence dependent and are not known until the vehicle route specified.

(7) All jobs are ready at the beginning. Initially, all parts are at the L/U station waiting for the vehicle to transport them to the machine which they should be processed.

(8) For each operation of all parts there are multiple machines can be choice. The same operation can be processed by different machines and the processing time is different. Different set of optional processing machines for different processed are different.

(9) Such issues as traffic control, congestion, machine failure or downtime, scraps, rework, and vehicle dispatches for battery change are ignored here and left as issues to be considered during real-time control

2.3 Objective Function

The object of problem is to minimize the makespan and the formulae used are given as follow:

Operation completion time:

$$\left(O_{ij}\right) = T_{ij} + P_{ij} \tag{1}$$

Job completion time:

$$\left(C_{ij}\right) = \sum_{i=1}^{n} O_{ij} \tag{2}$$

$$\text{Makespan} = \max\left(C_1, C_2, \dots, C_n\right) \tag{3}$$

Where j = operation, i = job, T_{ij}= travelling time, P_{ij} = operation process time

3 Proposed Algorithm

In this paper, we use GA and SA as the algorithm framework to design algorithm for solving the problem. The complete solutions which represent can be decomposed as three parts: job selection, machine selection and AGV selection. With combined the job selection and machine selection we can obtain the operation selection which determine how jobs processed on the machine. Then the transportation selection is to solve how to assign transportation mission to AGVs.

3.1 Encoding and Decoding

As stated before, the complete solution consists of three parts. Then we encode the feasible solution with the three-phase encoding frame. As figure 2 show that, the first phase represents the processing sequence of jobs, the second phase means the solution that machine selected for processing each operation of jobs; the last phase is to show the queue that transportation tasks assign on the AGVs.

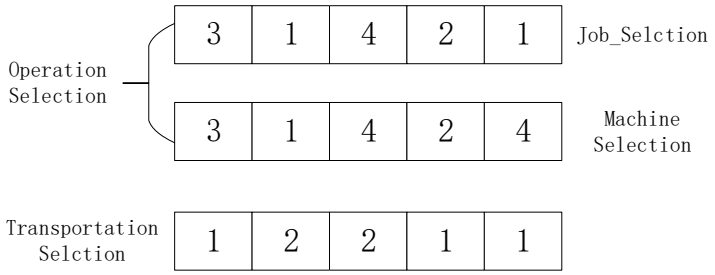


Figure 2 encoding example

Then we use the case which workshop has 3 jobs, 3 machines and 2 AGVs to explain the encoding and decoding. The table 1 and table 2 show the move time between each position and the processing times of each job.

Table 1 Move time matrix for the example problem

	L/U	1	2	3
L/U	0	1	2	1
1	1	0	1	2
2	1	2	0	1
3	2	1	1	0

Table 2 The processing times of each job

		M1	M2	M3
Job_1	Operation_1	-	2	4
	Operation_2	6	4	3
	Operation_3	-	3	4
Job_2	Operation_1	1	2	-
	Operation_2	4	-	3
	Operation_3	2	-	-
Job_3	Operation_1	6	3	-
	Operation_2	-	2	5
	Operation_3	1	-	4

Figure 3 represent the feasible solution with the code 313123221-233211313-211212121, 313123221 in this code represent the job selection code, machine selection code is 233211313, and transportation selection code is 211212121.

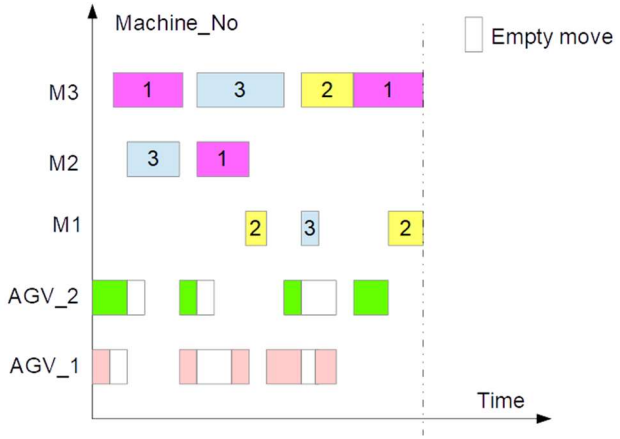


Figure 3 The Gantt chart for the example solution

3.2 Initialization

In scenes that need to solved quickly, a good initial solution can help the algorithm to get a better solution in a short time running time. We combine problems and greedy criterion to

get an initial solution. The greedy criterion also greedy from three phases. First, select the machine with the shortest process time for each operation of the job. Second, take the operation with the shortest processing time from the currently available operation. Last, use the earliest idle AGV to transport the current handling task.

3.3 Neighborhood structure for Algorithms

No matter the operator of GA's genetic operator or SA's neighborhood operator, it's neighborhood operator. And we both consider from operation selection and transportation selection. And the operation selection and transportation have a mutual effect.

From the perspective of AGV, the operation task sequence of each machine can be transformed to the transportation tasks. Then the AGVs handling these tasks can be seen as identical parallel machine scheduling problem. The neighborhood structure also can be design from operation selection and transportation selection respectively. Because there is no difference between the AGVs so we just only need change the AGVs for each transportation task and consider the symmetry attributes of this problem.

3.4 Repair Function

New solution generated by iteration does not necessarily obtain a legal solution. Illegal solution will cause the iteration of the solution to not work properly. For the illegal solution it should have a repair function.

The reason of an illegal solution is to select the machine that not available. Then we need to find the operation which is illegal and change an available machine for it.

3.5 Restart Scheme

Different restart scheme the algorithm choice will get different convergence speeds and diversity of solution. For GA, there are two kinds of typical scheme strategies: ranking and tournament, Ranking restart scheme can get higher diversity and the tournament restart scheme will get the fasted speed.

4 Experimental calibration of the algorithms

4.1 Algorithm parameter design

In this section, we study the behavior of the different operators of the proposed GAs and SAs. All combinations of the aforementioned factors yield many alternative GA. In order to calibration the algorithms, we have chosen a full factorial design in which all possible combinations of the following factors are tested[8].

Selection type for GA: 2 levels (Ranking and Tournament)

Crossover type for GA: 9 levels (SJOX, SJ2OX, SBOX, SB2OX, POS, OXⅡ, OXⅠ,CX,AP)

Mutation type for GA: 7 levels (SM, SIM, IVM, ISM, EM, DM)

Neighborhood operation for SA: 6 levels (SM, SIM, IVM, ISM, EM, DM)

4.2 Input Data

Every algorithm will be tested with four different kinds of layouts [4] in Figure 4. The distance from L/U point to machines and distance between a pair of machines are given in figure. The speed of the AGV is five distance units of time.

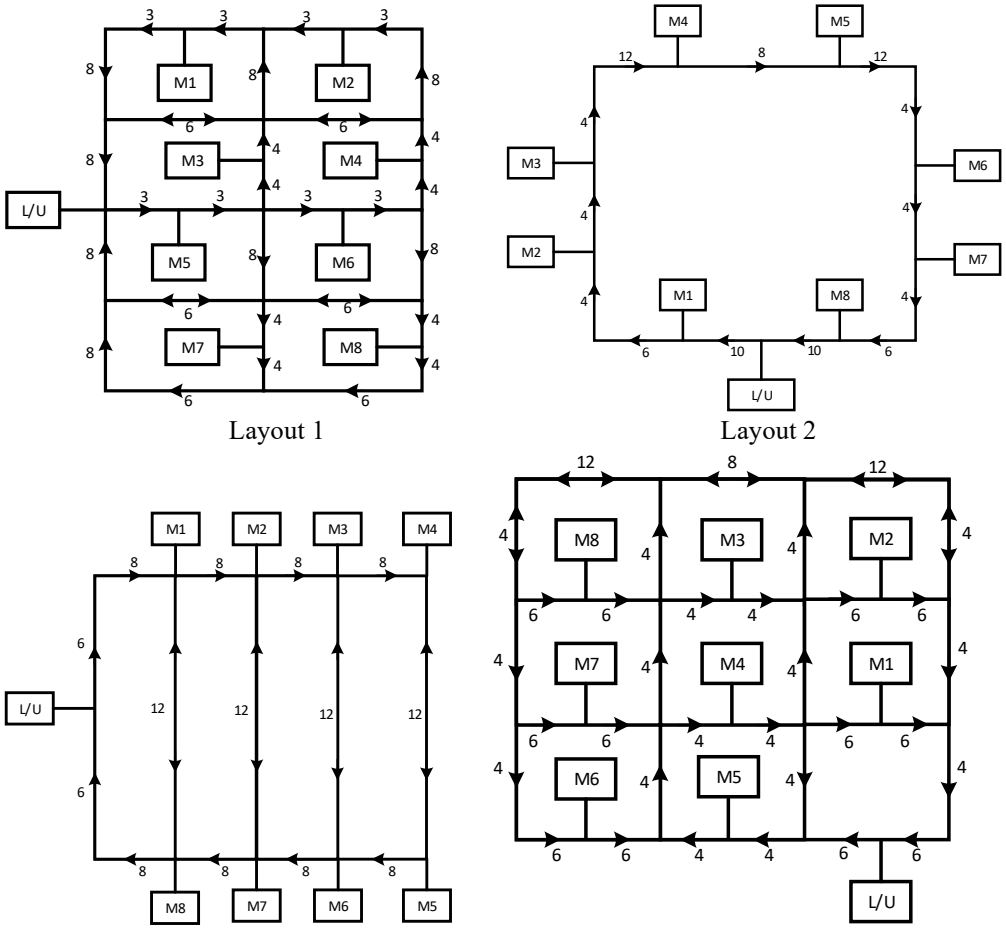


Figure 4 The layout configurations used as input data

Table 3 represents the collection of optional machining equipment for each operation. And the processing time corresponding to the machine for each job used in example problems in Table 4.

Table 3 Alternative operation matrix as input data

Job_No	Alternative operation and the machine no					
	Operation1	Operation2	Operation3	Opeation4	Operation5	Operation6
1	1,2,3	4,5,7	1,6,8	3,5,7	4,6,8	-
2	1,3,5	2,7	4,6,8	1,3,5	-	-
3	1,3	2,5	1,6,8	2,4	3,7	-
4	4,6,8	2,7	3,5	1,6	4,8	2,7

Table 4 The processing times of each job as input data

Job_No	Processing time of job for each operation					
	Operation1	Operation2	Operation3	Opeation4	Operation5	Operation6
1	6,7,10	5,57	6,7,4	5,7,7	4,8,1	-
2	8,5,6	5,6	6,9,5	9,10,11	-	-
3	5,6	7,7	10,9,7	12,9	7,9	-
4	5,7,8	8,6	10,11	13,9	12,9	13,10

Each algorithm is tested with the input data. The stopping criterion used when testing all instances with the algorithm is set to a CPU time which limit fixed to 1s. This stopping criterion set to be fixed because the impact of changes in the number of jobs and machines is not considered in these tests.

5 Computational experience

From the result of ANOVA analysis, by far, the greatest F-ratio corresponds to the non-controllable factor layout. This is clearly expected since difficult instances yield greater deviations from the lower bounds for the proposed algorithms. Since we cannot control the effect of layout (the parameter is input data) we focus on the following factors. The second biggest F-ratio value corresponds to the factor selection. The means plot for this factor can be observed in Figure 5. As we can see, there is a clear statistically significant different between Tournament and Ranking selection schemes and the former results in a better performance GA. As we will see from the rest of the statistical analysis, we can advance the fact that our proposed algorithms are robust to choose of operators.

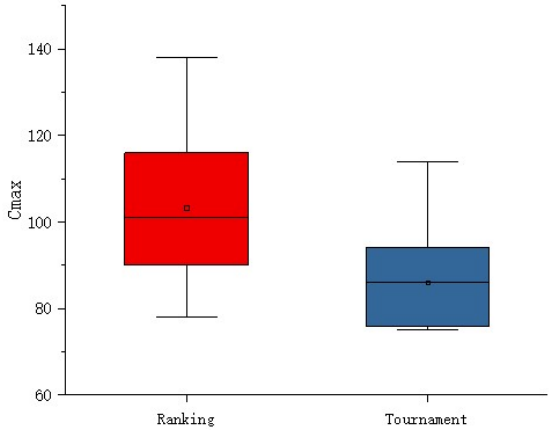
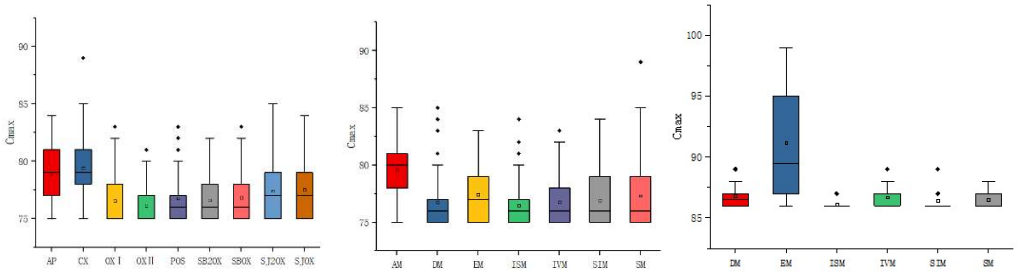


Figure 5 The box diagram for selection schemes

The next factor in order of importance are the type of mutation and crossover operators. Figure 6 show the corresponding means plot as box diagram. In can be seen that DM and ISM operator in mutation operation is clearly better than other operators. The OXⅡ operator is the best crossover operation. And the neighborhood operators of SA are IMS that we also can find in Figure 6.



a Crossover operators b Mutation operators c SA neighborhood

Figure 6 The box diagram for different operators

As the result of ANOVA and analyze the operation of mutation, crossover and neighborhood operation of SA, we get the best operation of in GA and SA. The table 5 shows the best result solved by GA, SA and Rules, we can see the result solved in a shortest running time GA and SA result has no different between each other and all better than the rules solutions.

Table 5 The best result for different layouts solved by each algorithm

Layout	Different kinds of Algorithm		
	GA	SA	Rule
Ex_1	75	75	87
Ex_2	76	76	90
Ex_3	92	92	104
Ex_4	86	86	98

6 Conclusions and future Research

In this work we have proposed two robust for the flexible job shop scheduling problem with limited transportation capacity under the makespan minimization criterion. We present 9 crossover operators that OXⅡ have shown the best performance. And SIM is the best not only in both of mutation operators but also in the set of SA neighborhood operators. And as the test example show that in a short run time GA and SA can get the same solution.

Furthermore, the GAs studied could be modified to take into account more realistic aspects of the problem such as the route that how AGVs move, different kinds of AGVs or consider the existence of due dates.

Acknowledgements This work is funded by Guangdong University of Technology. This project was supported by National Natural Science Foundation of China (Grant No.51775120, 51675107, 51975129, 71572049).

References

1. Sabuncuoglu I, Hommertzheim D, Dynamic dispatching algorithm for scheduling machines and automated guided vehicles in a flexible manufacturing system. International Journal of Production Research, 30(5), 1059-1079(1992).
2. Sigrid Knust, Shop-scheduling problems with transportation, Ph.D. Thesis, Fachbereich Mathematik / Informatik Universitat Osnabruck 1999
3. Rossi Andrea, Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships, International Journal of Production Economics, 153:253-267(2014).
4. Ulusoy G, Simultaneous scheduling of machines and automated guided vehicle, International Journal of Production Research, 31(12):2857-2873(1993).
5. He Y, Stecke K E, Smith M L, Robot and machine scheduling with state-dependent part input sequencing in flexible manufacturing systems, International Journal of Production Research,1-11(2016).
6. Mohammad S M, Saeed D A, Farshid E, Vahid M, An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs, Computers & Industrial Engineering, 86:2-13(2015).
7. Zheng Y, Xiao Y, Seo Y. A tabu search algorithm for simultaneous machine/AGV scheduling problem. International Journal of Production Research, 52(19):5748-5763(2014).
8. Rubén R, Concepción M, Alcaraz J . Two new robust genetic algorithms for the flowshop scheduling problem[J]. Omega, 34(5):461-476(2009).

MISTA 2019

Scheduling problems with assembly and delivery operations in Alibaba's Hema Store

Weiya Zhong · Hui Ding · Yaxin Yu

Abstract In this paper, we consider a scheduling model with assembly operation and delivery coordination, which is observed in Alibaba's Hema Store. Jobs are first processed in a machine environment with assembly machines, then delivered to customers located in different places. We formulate this problem into a mathematical model, study the time computational complexity, propose heuristic algorithms and carry out numerical experiments to reflect the performances and applicable conditions of the heuristic algorithms.
Keywords: Scheduling; Assembly; Delivery; Algorithm

1 Introduction

Alibaba′s Hema Store, the most popular new retail store in China, is a combination of supermarket, restaurant and vegetable market. A customer can also place an order in the Hema App and experience fast delivery: within 3 kilometers of the store, 30 minutes delivery to the door. Fast delivery is the biggest advantage of Hema. As soon as an order is placed in the Hema App, it is split into some suborders (according to the types of the goods in this order) by the information processing center. The goods of each suborder are first picked out by one staff in a specific area, then they are put into one bag and transmitted by a conveyer belt to the packing area. In the packing area, the bags for the same order are merged together and passed to the deliverymen. There are multiple orders in each delivery of a deliveryman, who will deliver the orders one by one to their customers.

We may formulate the orders processing placed in Hema APP as a machine scheduling problem with assembly and delivery operations. At the beginning of a planning horizon, n customers ordered n jobs $\{J_1, J_2, \cdots, J_n\}$. For $j = 1, 2, \cdots, n$, job J_j is composed of k sub-jobs $J_{j1}, J_{j2}, \cdots, J_{jk}$, where each sub-job J_{jl}, $l = 1, 2, \cdots, k$ is first processed non-preemptively by a dedicated two-stage flowshop machine (Denoted as $F_l(1, 1)$). The processing time of the two operations in the first and second stage of the flowshop of J_{jl} is p^1_{jl} and p^2_{jl}, respectively. For $j = 1, 2, \cdots, n$, after all the sub-jobs being processed in the two-stage flowshop machines, $J_{j1}, J_{j2}, \cdots, J_{jk}$ are assembled by a machine in the assembly stage with assembly time p^3_j. It is

Weiya Zhong
School of Management, Shanghai University
E-mail: wyzhong@shu.edu.cn

Hui Ding
School of Management, Shanghai University
E-mail: huilice@shu.edu.cn

Yaxin Yu
School of Management, Shanghai University
E-mail: 17839219692@qq.com

assumed that no machine can handle more than one task at a time. Now the jobs come to the delivery department where several jobs can be delivered in the same batch of a vehicle, under the constraint that the total number of the jobs is no more than B. The geographical relation of Hema Store and the customers is represented by a complete undirected graph $G = (V, E)$ of $n+1$ nodes $\{v_0, v_1, v_2, \cdots, v_n\}$ (node v_0 represents Hema Store and v_i represents the customer who orders job J_i). For $0 \leq g, h \leq n$ and $g \neq h$, there is an arc connecting v_g and v_h, and the length of the arc is t_{gh}, defined as the transport time between v_g and v_h. There are H vehicles for delivering the orders and each vehicle is first located at the machines (Hema Store). When one vehicle finishes delivering one batch of jobs, it returns from the last visited customer to the machines and can immediately deliver the un-delivered batch of jobs if there is any. Note that each customer should only be visited once.The objective is to minimize the makespan, defined as the time when the last vehicle finishes delivering all the jobs and returns to the machines.

What we need to do is: (1) to arrange the jobs' processing with assembly operations on the machines; (2) to decide the number of delivery batches and the jobs in each batch; (3) the delivery of each batch including the departure time from the machines and the exact route for delivering each job to its customer.

2 Literature Review

In this paper, we consider a scheduling problem with assembly and delivery operations. In the past years, there have been many papers published related to our problem, which may be divided into two categories: scheduling problems with assembly and scheduling problems with delivery.

Lee et al.([11]) first introduce makespan scheduling problem in the 3-machine (Denoted as Machine 1, 2, and 3) assembly-type flowshop, where each job contains three component tasks (Denoted as Task a, b, and c, respectively). Task $a/b/c$ is processed by Machine $1/2/3$, respectively, while task c cannot begin until task a and b have been completed. The authors prove that this problem is strongly NP-hard, present algorithms for a few polynomially solvable cases and develop approximate heuristics for the general case. Potts et al. ([16]) extend the problem in ([11]) to the case with multiple machines before the assembly stage, and propose approximation heuristics with worst-case analysis. Komaki et al. ([10]) consider two-stage hybrid flowshop makespan scheduling problems followed by single assembly machine, in which parts are manufactured in the flowshop area including multiple machines and each product is produced by assembly operation joining the parts. They provide better lower bounds and better heuristic algorithms with respect to the existent algorithms in the literature. Hatami et al. ([8]) study makespan problems with distributed flowshops and assembly operations. They propose a mathematical model, constructive algorithms and variable neighbourhood descent based algorithms. Zhang et al. ([20]) extend the problem in ([8]) by adding set-up time and multiple machines in the assembly stage. They propose a constructive heuristic and two hybrid metaheuristics and carry out numerical experiments to validate the algorithmic feasibility and effectiveness. Lin et al. ([14]) address the three-machine assembly-type flowshop scheduling problem with batching considerations to minimize the makespan. They polynomially solve a special case and study several heuristic algorithms to generate quality approximate solutions for the general problem.

Regarding the research on scheduling with delivery, there have been a large number of papers published on various academic journals. Hall and Potts ([7]) are among the first to consider scheduling problems with delivery coordination. Chen ([2]) gives an extensive review on the integrated production and outbound distribution scheduling problems before 2008, classifying models into five categories according to the configurations of production and outbound transportation. For integrated scheduling and delivery problems without routing decisions, Li et al. ([13]) and Toptal et al. ([17]) consider job families in production, where only jobs from the same job family can be transported in one batch. W. Zhong et al.([22]) and Fu et al. ([4]) assume that transportation is outsourced to third-party-logistics (3PL) providers, who picks up the jobs and deliver them to the customers subject to customers' requirements. For integrated scheduling and delivery problems without routing decisions, Ullrich ([18]) considers an integrated parallel machine scheduling and vehicle routing problem with time windows capacitated vehicles. The objective is to minimize total tardiness. Chang et al. ([1]) addresses an integrated parallel machine scheduling and vehicle routing problem without time window constraints. The objective is to minimize the weighted total job completion time and

total delivery cost. Zou et al. ([21]) study an integrated production scheduling and vehicle routing problem with a single machine for production and a limited number of vehicles with capacity constraints for transportation. The objective is to minimize the maximum order delivery time. Backward and forward batching methods are developed and embedded into a genetic algorithm, which is capable of providing high-quality solutions.

There is few research closely related to the scheduling problem in this paper. Li and Vairaktarakis ([12]) consider an integrated scheduling and distribution model in which jobs completed by two different machines must be bundled together for delivery. They assume that the bundling operation do not cost any time and the jobs can be delivered immediately after both parts were completed. The objective is to minimize the sum of the delivery and customers' waiting cost. For cases with only direct shipments and with milk-run deliveries, the authors develop polynomial-time heuristics and approximation schemes. Wang et al. ([19]) extend the model in ([11]) by considering the transportation of jobs to one customer. Jobs are transported in batches and each batch occurs a fixed delivery cost. Two simple heuristics and a novel hybrid meta-heuristic are presented to obtain near-optimal solutions.

Note that for the problem in this paper, if there is no delivery operation, a job's processing is completed after the sub-jobs are processed on the dedicated flowshops and are assembled. To the best of our knowledge, this scheduling model has not been proposed in any paper. Therefore, our scheduling problem observed in Alibaba's Hema store, with dedicated flowshops, assembly operations and delivery to customers, is also very interesting from the theoretical view.

In section 2, we study the case when each jobs contains two sub-jobs, there is a single machine for the assembly operations and one vehicle for delivering jobs. Computational time complexity is analyzed, and a mathematical programming model is formulated. In section 3, heuristic algorithms are proposed for the case in Section 2 and computational experiment is conducted to evaluate the performance of the algorithms. In section 4, we conclude this paper.

3 $F(1,1)(2) \rightarrow Ass(1) \rightarrow V(1)||D_{max}$

In this section, we study the scheduling model where each job is composed of two sub-jobs, the assembly operation is done by one single machine and one vehicle is initially located at the machines for delivering jobs. Following the classical three-field denotation of scheduling problem([6]), let the problem be $F(1,1)(2) \rightarrow Ass(1) \rightarrow V(1)||D_{max}$, where $F(1,1)(2)$ represents that each job is composed of two sub-jobs processed by two dedicated flowshop machines; $Ass(1)$ represents that there is a single machine for the assembly operations; $V(1)$ represents that there is one vehicle for the delivery and D_{max} represents that the objective is to minimize the time when the vehicle complets the delivery and returns to the machines.

3.1 Time computational complexity

Let us consider a special case of $F(1,1)(2) \rightarrow Ass(1) \rightarrow V(1)||D_{max}$, where for each sub-job J_{j2}, $j = 1,2,\cdots,n$, its processing time $p_{j2} = 0$ and transportation time $t_{gh} = 0$ for $g,h = 1,2,\cdots,n$ and $g \neq h$. Actually, this special case is a makespan in the 3-machine assembly-type flowshop scheduling problem and has been proved to be strongly NP-hard by Lee et al. [11]. Therefore, it is obvious that $F(1,1)(2) \rightarrow Ass(1) \rightarrow V(1)||D_{max}$ is also strongly NP-hard.

3.2 Mathematical formulation

Before the mathematical formulation is presented, the parameters and variables are defined as follows:

Parameters

i, j	index of the jobs (customers), which belongs to $\{1, 2, \cdots, n\}$
J_j	job J_j, which belongs to the job set $\{J_1, J_2, \cdots, J_n\}$
J_{j1} (J_{j2})	the first(second) sub-job of job J_j
$F_1(1,1)$ $(F_2(1,1))$	the two-stage flowshop machine for processing sub-job J_{j1} (J_{j2})
p_{j1}^1 (p_{j2}^1)	processing time of J_{j1} (J_{j2})on the first-stage machine of $F_1(1,1)$ $(F_2(1,1))$
p_{j1}^2 (p_{j2}^2)	processing time of J_{j1} (J_{j2})on the second-stage machine of $F_1(1,1)$ $(F_2(1,1))$
p_j^3	processing time of job J_j's assembly operation
$V = \{v_0, v_1, \cdots, v_n\}$	the nodes in the delivery graph
v_0	the node representing the Hema Store in the delivery graph
$v_1, v_2, \cdots, v_n$	the nodes representing n customers in the delivery graph
t_{gh}	transportation time between v_g and v_h, $g, h \in \{0, 1, \cdots, n\}$, $g \neq h$
B	capacity of the vehicle, i.e. maximal number of jobs in one delivery batch

Variables

s_{j1}^1 (s_{j2}^1)	starting time of sub-job J_{j1} (J_{j2}) on the first-stage machine of $F_1(1,1)$ $(F_2(1,1))$
c_{j1}^1 (c_{j2}^1)	finishing time of sub-job J_{j1} (J_{j2}) on the first-stage machine of $F_1(1,1)$ $(F_2(1,1))$
s_{j1}^2 (s_{j2}^2)	starting time of sub-job J_{j1} (J_{j2}) on the second-stage machine of $F_1(1,1)$ $(F_2(1,1))$
c_{j1}^2 (c_{j2}^2)	finishing time of sub-job J_{j1} (J_{j2}) on the second-stage machine of $F_1(1,1)$ $(F_2(1,1))$
s_j^3	starting time of job J_j on the assembly machine
c_j^3	finishing time of job J_j on the assembly machine
d	$\lceil \frac{n}{B} \rceil \leq d \leq n$, number of delivery batches
D_y	the $y-$th delivery batch, $y = 1, 2, \cdots, d$
s_{D_y}	departure time of D_y, $y = 1, 2, \cdots, d$
c_{D_y}	returning time of the vehicle after it finishes the jobs in D_y, $y = 1, 2, \cdots, d$
x_{jy}	$\begin{cases} = 1, & \text{if job } J_j \text{ is delivered in batch } D_y, \\ = 1, & \text{else.} \end{cases}$ $j \in \{1, 2, \cdots, n\}$, $y \in \{1, 2, \cdots, d\}$
	For ease of presentation, let $x_{0y} = 1$, $y \in \{1, 2, \cdots, d\}$.
e_{ij}	$\begin{cases} = 1, & \text{if the vehicle travels from } v_i \text{ to } v_j \text{ in some delivery batch,} \\ = 0, & \text{else.} \end{cases}$ $i, j \in \{0, 1, \cdots, n\}, i \neq j$
	For ease of presentation, let $e_{jj} = 0$, $j \in \{0, 1, \cdots, n\}$.

$$\min \quad c_{D_d} \tag{1}$$

$$s.t. \quad c_{jl}^q = s_{jl}^q + p_{jl}^q \qquad j \in \{1,2,\cdots,n\}, l \in \{1,2\}, q \in \{1,2\} \tag{2}$$

$$s_{jl}^2 \geq c_{jl}^1, \qquad j \in \{1,2,\cdots,n\}, l \in \{1,2\} \tag{3}$$

$$s_j^3 \geq \max\{c_{j1}^2, c_{j2}^2\}, \qquad j \in \{1,2,\cdots,n\} \tag{4}$$

$$c_j^3 = s_j^3 + p_j^3, \qquad j \in \{1,2,\cdots,n\} \tag{5}$$

$$(s_{il}^q - c_{jl}^q) \times (c_{il}^q - s_{jl}^q) \geq 0, \qquad i,j \in \{1,2,\cdots,n\}, i \neq j, q,l \in \{1,2\} \tag{6}$$

$$(s_i^3 - c_j^3) \times (c_i^3 - s_j^3) \geq 0, \qquad i,j \in \{1,2,\cdots,n\}, i \neq j \tag{7}$$

$$\sum_{j=1}^{n} x_{jy} \leq B, \qquad y \in \{1,2,\cdots,d\} \tag{8}$$

$$\sum_{y=1}^{d} x_{jy} = 1, \qquad j \in \{1,2,\cdots,n\} \tag{9}$$

$$s_{D_y} \geq \max_{j \in \{1,2,\cdots,n\}} x_{jy} c_j^3 \qquad y \in \{1,2,\cdots,d\} \tag{10}$$

$$\sum_{j=0}^{n} e_{ij} = 1, \qquad i \in \{1,2,\cdots,n\}, \tag{11}$$

$$\sum_{i=0}^{n} e_{ij} = 1, \qquad j \in \{1,2,\cdots,n\}, \tag{12}$$

$$\sum_{j=1}^{n} e_{0j} = d, \sum_{j=1}^{n} e_{j0} = d, \qquad j \in \{1,2,\cdots,n\}, \tag{13}$$

$$e_{ij}x_{jy} = e_{ij}x_{iy}, \qquad i,j \in \{1,2,\cdots,n\}, i \neq j, y \in \{1,2,\cdots,d\} \tag{14}$$

$$e_{ji}x_{jy} = e_{ji}x_{iy}, \qquad i,j \in \{1,2,\cdots,n\}, i \neq j, y \in \{1,2,\cdots,d\} \tag{15}$$

$$c_{D_y} = s_{D_y} + \sum_{i=0}^{n}\sum_{j=0}^{n} e_{ij}x_{iy}x_{jy}t_{ij}, \qquad y \in \{1,2,\cdots,d\} \tag{16}$$

$$s_{D_y} \geq c_{D_{y-1}} \qquad y \in \{2,3,\cdots,d\} \tag{17}$$

$$\lceil \frac{n}{B} \rceil \leq d \leq n, \qquad d \in \{\lceil \frac{n}{B} \rceil, \lceil \frac{n}{B} \rceil + 1, \cdots, n\} \tag{18}$$

$$x_{jy} \in \{0,1\}, e_{ij} \in \{0,1\}, \qquad i,j \in \{1,2,\cdots,n\}, y \in \{1,2,\cdots,d\} \tag{19}$$

Objective (1) is to minimize the time when the vehicle finishes delivering the last batch and returns to the machines. Constraints (2) and (5)ensure that on each machine, every job's completion time equals to its starting time plus processing time. Constraints (3)ensure that each sub-job's starting time on the second-stage flowshop machine is larger than its finishing time on the first-stage flowshop machine. Constraints (4) ensure that each job's assembly operation can be started only if its two sub-jobs have been finished production. Constraints (6) and (7) ensure that each machine can only process one job at one moment. Constraints (8) ensure that the number of jobs in one delivery batch do not exceed the capacity constraint.

Constraints (9) ensure that each job is delivered by one batch. Constraints (10) ensure that a delivery batch can be started only if all the jobs in this batch have been finished production. Constraints (11) and (12 ensure that each customer is visited exactly once. Constraints (13) ensure that there are exactly d batches delivered from the machines and the vehicle return to the machines d times. Constraints (14) and (15) ensure that if the vehicle travels between two customers, then the two customer's jobs must be delivered in one batch. Constraints (16) ensures that for each delivery, the time when the vehicle returns to the machines equal to the starting time plus the time travelling between all the nodes in this batch. Constraints (17) ensures that for each delivery, the vehicle can start only if it has returned after delivering the jobs in the last batch. Constraints (18) ensures that the number of batches is in the right range. Constraints (19) are integer conditions.

3.3 Heuristic algorithms

Duo to the strongly NP-hardness of $F(1,1)(2) \rightarrow Ass(1) \rightarrow V(1)||D_{max}$, it is impossible to get optimal solution in reasonable time. Therefore, heuristic algorithms are developed to solve this problem, which jointly consider the machine scheduling part and the delivery operation, including the jobs' production sequence, the jobs' batching decision and the routing decision for each delivery batch. We will apply Johnson's rule([15]) for jobs' production sequence and genetic algorithms (GA) for the routing decision in each delivery batch. Genetic algorithms ([9], [5] and [3]) are stochastic search methods which are part of the evolution-motivated meta-heuristics.

In the following, we first list a property for the optimal solution.

Property 1 There exist an optimal solution for $F(1,1)(2) \rightarrow Ass(1) \rightarrow V(1)||D_{max}$, in which the jobs in the same delivery batch are processed consecutively on the assembly machine.

The proof is quite simple and we omit it.

Genetic algorithm BVR

Recall that the geographical relation of Hema Store and the customers is represented by a complete undirected graph $G = (V, E)$ of $n+1$ nodes $\{v_0, v_1, v_2, \cdots, v_n\}$ (node v_0 represents Hema Store and v_i represents the customer who orders job J_i).

1. Coding and decoding of chromosomes
   A chromosome represents a permutation of the nodes visited in one batch, including the assembly machine v_0 and some customers. For example, suppose that one batch contains four jobs from four customers, represented by v_1, v_2, v_3 and v_4, respectively. $(1, 2, 3, 4)$ is a chromosome, meaning that the vehicle starts from v_0, first goes to v_1, followed by v_2, then goes to v_3, followed by v_4, at last, the vehicle returns to v_0. The chromosomes in the initial population are generated in a random method, where each node is inserted randomly into a chromosome. Each chromosome is an individual in the population, and can be converted into a feasible solution of the vehicle routing decision of one batch.
2. Genetic operators
   The genetic operators include selection, crossover, mutation, reverse and reinsert. In general, the parents are selected first, then the crossover operator is applied to produce offspring. Thirdly, the mutation operator is executed on the generated offspring for the purpose of diversification. Then, in order to improve the local search ability, reverse operator is applied. Lastly, reinsert operator is executed to solve the chromosome reduction due to selection operator.

Batching methord BM

1. Divide customer nodes$\{v_1, v_2, \cdots, v_n\}$ into $d = \lceil \frac{n}{B} \rceil$ subsets randomly, where one subset contains $n - B\lceil \frac{n}{B} \rceil$ nodes and each of the other $\lceil \frac{n}{B} \rceil - 1$ subsets contains B nodes. Denote the batches be $\mathcal{B}_y$, $y = 1, 2, \cdots, d$.
2. For each batch $\mathcal{B}_y$, apply genetic algorithm BVR to obtain a routing decision which starts from v_0, visits the customers in this batch one by one and returns to v_0, and let the total routing time be $T(\mathcal{B}_y)$.

3. Compute the total routing time of the batches' routing time $\sum_{y=1}^{d} T(\mathcal{B}_y)$.
4. Repeat the above three steps M times (M is a positive integer) and select the batching rule with the smallest $\sum_{y=1}^{d} T(\mathcal{B}_y)$.

Heuristic H_{MD1}

1. For $l = 1, 2$, process sub-jobs $J_{1l}, J_{2l}, \cdots, J_{nl}$ on flowshop machines $F_l(1,1)$ according to Johnson's rule [15], where sub-job J_{jl}'s processing time on the first-stage(second-stage) machine is $p_{jl}^1(p_{jl}^2)$. Let the completion time of sub-job J_{jl}^2 be c_{jl}^2, $l = 1, 2$ and $c_j^2 = \max\{c_{j1}^2, c_{j2}^2\}$, $j = 1, 2, \cdots, n$.
2. Sequence the jobs in non-decreasing order of c_j^2, and w.o.l.g. let the jobs' sequence be $\{J_1, J_2, \cdots, J_n\}$. Process the jobs in this order on the assembly machine one by one.
3. Jobs are divided into $d = \lceil \frac{n}{B} \rceil$ delivery batches, denoted as $\mathcal{B}_y$, where the $y-$th batch contains jobs $J_{(y-1)B}, J_{(y-1)B+1}, \cdots, J_{yB}$, for $y = 1, 2, \cdots, d-1$, and the last delivery batch contains jobs $J_{yB+1}, J_{yB+2}, \cdots, J_n$. When job J_B is completed on the assembly machine, the vehicle starts the first delivery batch $\mathcal{B}_1$ and the vehicle routing decision is made by genetic algorithm BGV. When the vehicle returns to the assembly machine after all the jobs in $\mathcal{B}_1$ have been delivered to their customers, if job J_{2B} have been completed on the assembly machine, then the vehicle starts the second delivery batch $\mathcal{B}_2$, else, the vehicle will wait until job J_{2B} is completed on the assembly machine. The vehicle routing decision is made by genetic algorithm BGV. Following this way, the vehicle delivers all the jobs to their customers and returns to the assembly machine at last.

Let the objective function value obtained from H_{MD1} be $C_{H_{MD1}}$.

Heuristic H_{MD2}

1. Let $P_1 = \sum_{j=1}^{n}(p_{j1}^1 + p_{j1}^2)$ and $P_2 = \sum_{j=1}^{n}(p_{j2}^1 + p_{j2}^2)$. W.o.l.g., assume that $P_1 \leq P_2$. Process the sub-jobs on $F_1(1,1)$ and $F_2(1,1)$ in the same order, which is generated from Johnson's rule for sub-jobs J_{j1}, where sub-job J_{j1}'s processing time on the first-stage(second-stage) machine is $p_{j1}^1(p_{j1}^2)$. Let the completion time of sub-job J_{jl}^2 be c_{jl}^2, $l = 1, 2$ and $c_j^2 = \max\{c_{j1}^2, c_{j2}^2\}$, $j = 1, 2, \cdots, n$.
2. Same as step 2 of H_{MD1}.
3. Same as step 3 of H_{MD1}.

Let the objective function value obtained from H_{MD2} be $C_{H_{MD2}}$.

Heuristic H_{MD3}

1. Let $P_1 = \sum_{j=1}^{n} p_{j1}^1 + p_{j1}^2$ and $P_2 = \sum_{j=1}^{n} p_{j2}^1 + p_{j2}^2$. W.o.l.g, assume that $P_1 \leq P_2$. Process the sub-jobs on $F_1(1,1)$ and $F_2(1,1)$ in the same order, which is generated from Johnson's rule for sub-jobs J_{j2}, where sub-job J_{j2}'s processing time on the first-stage(second-stage) machine is $p_{j2}^1(p_{j2}^2)$. Let the completion time of sub-job J_{jl}^2 be c_{jl}^2, $l = 1, 2$ and $c_j^2 = \max\{c_{j1}^2, c_{j2}^2\}$, $j = 1, 2, \cdots, n$.
2. Same as step 2 of H_{MD1}.
3. Same as step 3 of H_{MD1}.

Let the objective function value obtained from H_{MD3} be $C_{H_{MD3}}$.

Heuristic H_{DM1}

1. Run batching method BM to get the delivery batches and the routings inside each batch, and let $\mathcal{B}_y$ be the delivery batch and let $T(\mathcal{B}_y)$ be the running time of the routing for delivering jobs in $\mathcal{B}_y$, $y = 1, 2, \cdots, d$, $d = \lceil \frac{n}{B} \rceil$. Sequence the batches in non-decreasing order of $T(\mathcal{B}_y)$, and w.o.l.g. assume the batches' sequence be $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_d$.
2. The processing sequences of sub-jobs on flowshops $F_1(1,1)$ and $F_2(1,1)$ are the same. Sub-jobs in batch $\mathcal{B}_y$ are processed prior to jobs in $\mathcal{B}_{y+1}$, $y = 1, 2, \cdots, d-1$. For each batch $\mathcal{B}_y$, process the sub-jobs on $F_1(1,1)$ ($F_2(1,1)$) in the same order as the delivery order obtained in $\mathcal{B}_y$ and assemble the jobs on the assembly machine in this order, too.

3. When jobs in $\mathcal{B}_1$ is completed on the assembly machine, the vehicle starts the first delivery batch $\mathcal{B}_1$ and the vehicle routing decision is made by BM. When the vehicle returns to the assembly machine after all the jobs in $\mathcal{B}_1$ have been delivered to their customers, if jobs in $\mathcal{B}_2$ have been completed on the assembly machine, then the vehicle starts the second delivery batch $\mathcal{B}_2$, else, the vehicle will wait until jobs in $\mathcal{B}_2$ are completed on the assembly machine. The vehicle routing decision is made by BM. Following this way, the vehicle delivers all the jobs to their customers and returns to the assembly machine at last.

Let the objective function value obtained from H_{DM1} be $C_{H_{DM1}}$.

Heuristic H_{DM2}

1. Same as step 1 of H_{DM1}.
2. Sub-jobs in batch $\mathcal{B}_y$ are processed prior to jobs in $\mathcal{B}_{y+1}$, $y = 1, 2, \cdots, d-1$. For each batch $\mathcal{B}_y$, process the sub-jobs on $F_l(1,1)$ according to Johnson's rule, where sub-job J_{jl}'s processing time on the first-stage(second-stage) machine is $p^1_{jl}(p^2_{jl})$, $l = 1, 2$. Let the completion time of sub-job J^2_{jl} be c^2_{jl}, $l = 1, 2$ and $c^2_j = \max\{c^2_{j1}, c^2_{j2}\}$. Assemble the jobs in $\mathcal{B}_y$ in non-decreasing order of c^2_j.
3. Same as step 3 of H_{DM1}.

Let the objective function value obtained from H_{DM2} be $C_{H_{DM2}}$.

Heuristic H_{DM3}

1. Same as step 1 of H_{DM1}.
2. The processing sequences of sub-jobs on flowshops $F_1(1,1)$ and $F_2(1,1)$ are the same. Sub-jobs in batch $\mathcal{B}_y$ are processed prior to jobs in $\mathcal{B}_{y+1}$, $y = 1, 2, \cdots, d-1$. Let $P_1(\mathcal{B}_y) = \sum_{J_j \in \mathcal{B}_y}(p^1_{j1} + p^2_{j1})$, $P_2(\mathcal{B}_y) = \sum_{J_j \in \mathcal{B}_y}(p^1_{j2} + p^2_{j2})$ and $o_y = argmax\{P_1(\mathcal{B}_y), P_2(\mathcal{B}_y)\}$. For each batch $\mathcal{B}_y$, process the sub-jobs on $F_1(1,1)$ and $F_2(1,1)$ in the same order, which is generated from Johnson's rule for sub-jobs J_{jo_y}, where sub-job J_{jo_y}'s processing time on the first-stage(second-stage) machine is $p^1_{jo_y}(p^2_{jo_y})$. Assemble the jobs on the assembly machine in this order, too.
3. Same as step 3 of H_{DM1}.

Let the objective function value obtained from H_{DM3} be $C_{H_{DM3}}$.

Heuristic H_{DM4}

1. Same as step 1 of H_{DM3}.
2. Same as step 1 of H_{DM3} except that let $o_y = argmin\{P_1(\mathcal{B}_y), P_2(\mathcal{B}_y)\}$
3. Same as step 3 of H_{DM3}.

Let the objective function value obtained from H_{DM4} be $C_{H_{DM4}}$.

3.4 Computational Experiment

As we know that, the scheduling problem in this paper is rather complex due to that multiple factors are involved, including the processing of the sub-jobs, assembly operations, batching decisions and routing decisions of the vehicle. In order to obtain efficient solutions with high quality, we developed various heuristic algorithms in the previous subsection. In this subsection, we carry out computational experiment to evaluate the performance of these heuristic algorithms. Computational experiment in this paper is carried out by Matlab 2017b in the system of Inter Core i7-4600U 2.1GHz 8G RAM.

1. Data set generation

For heuristic algorithms in subsection 3.3, batching decisions and routing decisions in H_{MD1}, H_{MD2} and H_{MD3} are generated based on jobs' processing sequence on the machines while jobs' processing sequence in H_{DM1}, H_{DM2}, H_{DM3} and H_{DM4} is determined according to the batching decisions and routing decisions. For various combinations of problem parameters, including jobs' processing times at each stage, vehicle's capacity and transportation times between nodes in the delivery graph, multiple instance sets containing

Table 1: computational results for type-1 instances

Instance	H_{MD1}		H_{MD2}		H_{MD3}		H_{DM1}		H_{DM2}		H_{DM3}		H_{DM4}	
	O	T	O	T	O	T	O	T	O	T	O	T	O	T
1	2097	4.1	2039	4.4	2020	4.3	2345	17.4	2291	17.3	2237	17.6	2246	17.6
2	2108	4.3	2030	4.6	2037	4.7	2349	17.9	2297	17.6	2243	17.9	2247	17.6
3	2133	4.4	2050	4.7	2069	4.5	2376	17.6	2316	18.2	2257	17.9	2262	17.7
4	2037	3.6	1965	4.0	1977	4.0	2290	18.5	2232	16.4	2176	17.1	2170	18.1
5	2024	4.2	1955	4.3	1970	4.4	2261	18.2	2214	17.2	2167	18.3	2168	18.6
6	2120	4.1	2050	3.7	2034	3.8	2360	18.8	2312	18.4	2271	18.1	2262	18.4
7	2139	3.9	2086	4.4	2080	4.3	2388	16.8	2342	17.3	2292	17.9	2281	18.1
8	2199	4.3	2127	4.2	2110	4.4	2445	18.1	2393	17.5	2357	18.3	2347	18.4
9	2140	4.2	2070	4.3	2076	4.3	2383	18.2	2331	17.1	2280	18.6	2284	17.6
10	2149	4.1	2083	4.7	2079	4.2	2385	17.3	2334	16.7	2282	18.3	2277	17.2
Average	2115	4.1	2046	4.3	2045	4.3	2358	17.9	2306	17.4	2256	18.0	2254	17.9

100 jobs are produced to reflect the performances and applicable conditions of the heuristic algorithms. In the following tables, the first column displays the detailed parameters of each instance. Take column one line one in Table 1 for example, $B = 20$ means the vehicle's capacity is 20, $t \in [6, 10]$ means the travel times between two nodes in the delivery network are uniform distributed random integers in $[6, 10]$, $p_{j1}^1 \in [10, 30]$ means the processing time of sub-job J_{j1}'s first stage operations are uniform distributed random integers in $[10, 30]$. Similarly, $p_{j1}^2 \in [8, 20]$, $p_{j2}^1 \in [10, 30]$, $p_{j2}^2 \in [8, 20]$ and $p_{j3} \in [5, 6]$ means parameters p_{j1}^2, p_{j2}^1, p_{j2}^2 and p_{j3} are uniform distributed random integers in the corresponding intervals.

2. Parameters for GA When genetic algorithm BVR is applied, there are five parameters needed to be fixed, including population size PZ, number of iterations NI, crossover rate CR, mutation rateMR and generation gap $GGAP$. Since the effectiveness of the heuristic algorithms depends on the parameter values, algorithm H_{MD1} is run to pre-test various combinations of $(PZ, NI, CR, MR, GGAP)$ on some selected instances with $n = 100$ and $B = 20$. (Due to length constraint, we omit the details.)

3. Computational results and analysis

In Table 1, computational results are list for type-1 instances (with $q = 20$, $t \in [6, 10]$, $p_{j1}^1 \in [10, 30]$, $p_{j1}^2 \in [8, 20]$, $p_{j2}^1 \in [10, 30]$, $p_{j2}^2 \in [8, 20]$ and $p_{j3} \in [5, 6]$), where S means solution value and T means running time. For each instance, heuristic algorithms H_{DM1}, H_{DM2}, H_{DM3}, H_{DM4}, H_{MD1}, H_{MD2} and H_{MD3} are applied. The objective value(V) and running time (T) is recorded for each instance by each heuristic algorithm. In Tables 2 and 3, similar results are list for type-2 instances (with $q = 20$, $t \in [21, 40]$, $p_{j1}^1 \in [1, 8]$, $p_{j1}^2 \in [1, 6]$, $p_{j2}^1 \in [1, 8]$, $p_{j2}^2 \in [1, 6]$ and $p_{j3} \in [1, 2]$) and type-3 instances (with $q = 20$, $t \in [11, 20]$, $p_{j1}^1 \in [5, 15]$, $p_{j1}^2 \in [5, 10]$, $p_{j2}^1 \in [5, 15]$, $p_{j2}^2 \in [5, 10]$ and $p_{j3} \in [3, 4]$).

(1) For the above three types of instances, the running time of each heuristic algorithm is stable. It takes almost the same amount of time for H_{MD1}, H_{MD2} and H_{MD3} (Denote these three algorithms as MD-style algorithms), and the same is true for H_{DM4}, H_{MD1}, H_{MD2} and H_{MD3} (Denote these three algorithms as DM-style algorithms). Furthermore, MD-style algorithms run much faster than DM-style algorithms. This is because a DM-style algorithm first decides the routing decisions for the delivery and applies batching method "BM" as a sub-procedure. Batching method "BM" cost a lot of time since it tries many possible routings and selects the best one (Note that each routing is obtained by the genetic algorithm BVR).

(2) For each type-1 instance, the objective value obtained from one MD-style algorithm is less than that from any DM-style algorithm, while for each type-2 instance, the objective value obtained from one

Table 2: computational results for type-2 instances

Instance	H_{MD1}		H_{MD2}		H_{MD3}		H_{DM1}		H_{DM2}		H_{DM3}		H_{DM4}	
	O	T	O	T	O	T	O	T	O	T	O	T	O	T
1	2976	4.3	2819	4.7	2741	4.8	2661	18.1	2626	18.3	2521	17.9	2516	18.2
2	2957	4.9	2808	4.6	2802	4.9	2635	18.6	2620	18.4	2531	18.3	2544	18.4
3	2963	4.7	2747	4.3	2735	4.9	2688	18.0	2614	18.7	2487	18.4	2547	18.6
4	2885	3.7	2651	4.2	2639	3.9	2658	18.8	2582	18.0	2463	17.6	2501	18.7
5	2966	4.2	2756	4.3	2789	4.2	2678	17.3	2606	16.8	2542	18.1	2544	18.2
6	2943	4.3	2766	4.3	2792	4.3	2611	16.5	2598	16.9	2499	18.1	2494	18.3
7	2959	4.5	2778	4.4	2775	4.5	2653	18.1	2611	18.8	2511	18.9	2518	18.5
8	2927	3.9	2785	4.2	2753	4.6	2655	17.5	2570	17.3	2502	17.2	2495	19.2
9	2946	4.6	2732	4.5	2727	4.7	2649	18.6	2579	18.6	2490	18.9	2500	19.0
10	2907	4.9	2800	4.8	2734	4.8	2663	18.5	2601	19.0	2502	18.4	2484	18.6
Average	2943	4.4	2764	4.4	2749	4.6	2655	18.0	2601	18.1	2505	18.2	2514	18.6

Table 3: computational results for type-3 instances

Instance	H_{MD1}		H_{MD2}		H_{MD3}		H_{DM1}		H_{DM2}		H_{DM3}		H_{DM4}	
	O	T	O	T	O	T	O	T	O	T	O	T	O	T
1	1605	4.4	1487	4.4	1449	4.3	1553	18.6	1568	17.6	1413	17.5	1418	16.8
2	1565	4.6	1467	4.5	1464	4.5	1725	17.5	1485	17.7	1405	16.7	1421	17.3
3	1636	4.7	1463	4.5	1481	4.7	1539	17.9	1484	17.0	1421	18.0	1458	16.8
4	1515	4.6	1354	4.8	1346	4.2	1515	18.2	1526	17.4	1402	17.7	1395	17.4
5	1682	4.2	1483	4.8	1417	4.2	1530	18.1	1498	17.1	1404	17.3	1430	17.8
6	1625	3.9	1460	4.4	1425	4.1	1517	18.2	1510	17.5	1414	17.1	1414	17.6
7	1644	4.3	1449	4.2	1471	4.1	1749	17.3	1705	17.5	1641	17.5	1625	18.0
8	1468	3.7	1369	3.9	1382	4.0	1546	17.1	1496	17.5	1400	18.3	1392	18.0
9	1641	4.5	1477	4.1	1477	4.1	1536	18.0	1492	17.8	1432	17.8	1419	18.4
10	1616	4.3	1460	4.7	1449	4.3	1619	17.7	1507	17.8	1397	17.8	1390	17.3
Average	1600	4.3	1447	4.4	1436	4.3	1583	17.9	1527	17.5	1433	17.6	1436	17.5

MD-style algorithm is larger than that from any DM-style algorithm. It is also easy to see that, for type-3 instances there is no such obvious advantages or disadvantages between MD-style and DM-style algorithms.

The interpretation for the performances of various algorithms on different instances lies in the instances' parameter settings and the algorithm's design.

(2.1) For each type-1 instance, the processing time of a job's operation is much larger than the vehicle's transportation time between two nodes in the delivery network. Therefore, the idle time in the whole system (including the machine environment and the delivery part) before the last job is delivered and the vehicle returns to the machines is mainly caused in jobs' processing part and the first concern must be this part. For MD-style algorithms, Johnson's rule is first applied for sub-jobs' processing before the assembly stage, then jobs' batching decision and departure times of batches are generated on the basis of jobs' processing decision. For DM-style algorithms, jobs' batching decision is first made, then jobs' processing strategies

are obtained one batch by one batch. So it is reasonable that, for type-1 instances MD-type algorithms outperform DM-type algorithms.

(2.2) For each type-2 instance, the processing time of a job's operation is much smaller than the vehicle's transportation time between two nodes in the delivery network. Therefore, the idle time in the whole system is mainly caused by finished jobs' waiting for the vehicle and the first concern must be jobs' batching and routing decsions. By similar analysis as in (2.1), it is reasonable that for type-2 instances DM-type algorithms outperform MD-type algorithms.

(2.3) For each type-2 instance, the gap between the values of jobs' processing time and vehicle's transportation time narrows. Therefore, in table 3, there is no obvious difference between the objective values obtained from MD-type and DM-type algorithms.

(3) When MD-style algorithms are run for an instance (no matter which type it belongs to), there is no obvious difference between $C_{H_{MD2}}$ and $C_{H_{MD3}}$ (the objective values obtained from algorithms H_{MD2} and H_{MD3}), while $C_{H_{MD1}}$ (the objective value obtained from algorithm H_{MD1} is always larger than $C_{H_{MD2}}$ and $C_{H_{MD3}}$. Note that, in algorithm H_{MD1}, the sequence of sub-jobs on flowshops $F_1(1,1)$ and $F_2(1,1)$ follows different rules, which are generated by Johnson's rule separately. In algorithms H_{MD2} and H_{MD3}, the sequence of sub-jobs follows the same rule. This explains the weak performance of algorithm H_{MD1}, since a job's assembly operation can be started only if its two sub-jobs are completed. If the sub-jobs' sequences before the assembly stage are different, there will be more idle time in the whole system.

(4) When DM-style algorithms are run for an instance (no matter which type it belongs to), there is no obvious difference between $C_{H_{DM3}}$ and $C_{H_{MD4}}$. The performance of H_{MD2} is not as good as H_{MD3} or H_{MD4}, which can be explained by similar reasons as in (3), since in algorithm H_{DM2}, the sequence of sub-jobs on flowshops $F_1(1,1)$ and $F_2(1,1)$ are different. It is easy to see that the performance of H_{MD2} is better than H_{MD1}. For H_{MD1}, the jobs' sequence inside each batch is the same as the visiting sequence of the delivery, without considering the jobs' processing times.

In summary, above algorithms jointly consider jobs' processing and delivery. However, which is the prime concern, "processing" or "delivery", depends on the values of the instance's variables. Inside each delivery batch, it is better to process of sub-jobs on different flowshops in the same order.

4 Conclusion

In this paper, we consider a scheduling model with assembly operation and delivery coordination, which is observed in Alibaba's Hema Store. We formulate this problem as a mathematical model, study the time computational complexity, propose heuristic algorithms and carry out numerical experiments to reflect the performances and applicable conditions of the heuristic algorithms. Furthermore, the mathematical formulation and heuristic algorithms can be easily revised for the case with multiple vehicles.

For future research, there exists many extended scheduling problems worthy of future study and applicable significance, e.g. (1) cases with more than two types of sub-jobs, (2) cases with hybrid flowshops for the production of sub-jobs and (3) cases with multiple assembly machines. Designing fast and efficient heuristic algorithms is interesting and a big challenge.

References

1. Y.C. Chang, V.C. Li, C.J. Chiang, An ant colony optimization heuristic for an integrated production and distribution scheduling problem, Engineering Optimization, Volume 46(4), 503-520(2014).
2. Z.L. Chen, Integrated production and outbound distribution scheduling: review and extensions, Operations Research, Volume 58(1), 130-148(2010).
3. L. Davis, Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991).
4. B. Fu, Y. Huo, H. Zhao, Coordinated scheduling of production and delivery with production window and delivery capacity constraints, Theoretical Computer Science, Volume 422, 39-51(2012).
5. D.E. Goldberg, Genetic Algorithms in Search. Optimization, and Machine Learning. Addison-Wesley, Reading(1989).
6. R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, Volume 5, 278-326(1979).

7. N.g. Hall, and C.N.Potts, Supply chain scheduling: batching and delivery, Operations Research, Volume 51(4), 566-584(2003).
8. S. Hatami, R. Ruzi, and C. A. Romono, The distributed assembly permutation flowshop scheduling problem, International Journal of Production Research, Volume 51(17), 5292-5308(2013).
9. J.H. Holland, Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975).
10. G.M. Komaki, Ehsan Teymourian, Vahid Kayvanfar, Minimizing makespan in the two-stage assembly hybrid flow shop scheduling problem using artificial immune systems, International Journal of Production Research, Volume 54(4), 963-983(2016).
11. C.-Y. Lee, T.C.E. Cheng, B.M.T. Lin, Minimizing the makespan in the 3-Machine Assembly-type flowshop scheduling problem, Management Science, Volume 39(5), 616-625(1993).
12. C.-L. Li, G. Vairaktarakis, Coordinating production and distribution of jobs with bundling operations, IIE Transactions, Volume 39(2), 203-215(2007).
13. S. Li, J. Yuan, B. Fan, Unbounded parallel-batch scheduling with family jobs and delivery coordination, Information Processing Letters, Volume 111(12), 575-582(2011).
14. B.M.T. Lin, T.C.E. Cheng, A.S.C. Chou, Scheduling in an assembly-type production chain with batch transfer, OMEGA (The International Journal of Management Science), Volume 35(2), 143-151(2007).
15. S.M. Johnson, Optimal two- and three-stage produciton schedules with setup times included, Naval Research Logistics Quarterly, Volume 1, 61-68(1954).
16. C.N.Potts, S.V.Sevastjanov, V.A. Strusevich, L.N.V. Wassenhove, C.M.Zwaneveld, The two-stage assembly scheduling problem: complexity and approximation, Operations Research, Volume 43(2), 346-355(1995).
17. A. Toptal,, U. Koc, and I. Sabuncuoglu, A joint production and transportation planning problem with heterogeneous vehicles, International Journal of Production Research, Volume 65(2), 180-196(2014).
18. C.A. Ullrich, Integrated machine scheduling and vehicle routing with time windows, European Journal of Operational Research, Volume 227(1), 152-165(2013).
19. K. Wang, W.Q. Ma, H. Luo, and H. Qin, Coordinated scheduling of production and transportation in a two-stage assembly flowshop,International Journal of Production Research, Volume 54(22), 6891-6911(2016).
20. G. Zhang, K. Xing, and F. Cao, Scheduling distributed flowshops with flexible assembly and set-up time to minimize the makespan, International Journal of Production Research, Volume 56(9), 3226-3244(2018).
21. X. Zou, L. Liu, K. Li, and W. Li, A coordintated algorithm for integrated production scheduling and vehicle routing problem, International Journal of Production Research, Volume 56(15), 5005-5024(2018).
22. W. Zhong, Z.L. Chen, M. Chen, Integrated production and distribution scheduling with committed delivery dates , Operations Research Letters, Volume 38(2), 133-138(2010).

MISTA 2019

Optimization of Heat Exchanger Cleaning Schedule for Natural Gas Liquefaction Process

Hermawan Nugroho • Derma Wiharsa Kusuma • Totok Ruki Biyanto

Abstract In liquefaction of natural gas, heat exchanger is a very important component. Liquefaction involves the transfer of energy from hot stream of natural gas to cold stream of the refrigerant through heat exchangers. As time passes, the performance of heat exchanger decreases due to fouling. Fouling is defined as the accumulation of unwanted deposits on surfaces. They impose an additional resistance to heat and cannot be avoided. To reduce it, it needs periodic cleaning. Periodic cleaning can improve and restore the efficiency of the heat exchangers. However it should be noted that cleaning will also contribute in increasing cost. So the cleaning schedule needs to be optimized. In this project, we develop an optimized strategy of cleaning schedule. The proposed method is based on a stochastic approach knows as duelist algorithm. The proposed method is tested on a plant model that developed based on real plant. Results shows that the proposed method is able to save 446,264 MJ/s (heat duty), 681,919 MWh (pumps power) and $ 841,000 in the period 72 months.

1 Introduction

The need for energy is increasing worldwide including the need of natural gas. It is projected that it will increase to 497 Bcf per day in 2035 [1]. Most of the demands are coming from industry and power sector with 71 Bcf per day and 63 Bcf per day respectively with an

estimated of 1.9% increment per year. Because of its flexibility, liquefied natural gas (LNG) is the most demanded from natural gas market and will reach 46% in 2035. The supply will be dominated by North America countries with 70% in 2035 (reduced from 99% in 2014).

With increasing demand for LNG and the fact that energy plays a major role in industries, it will definitely influence production and processing costs. Industries will require a more efficient approach to reduce their costs.

LNG is natural gas that has been cooled to –260° F (–162° C), changing it from a gas into a liquid that is 1/600th of its original volume. In the liquefaction of natural gas, refrigerant gas is used to cool the natural gas. The temperature of the gas is eventually reduced to −161°C, the temperature at which methane, the main constituent of natural gas, liquefies. At this temperature, all the other hydrocarbons in the natural gas will also be in liquid form.

The refrigeration and liquefaction section is the key element of an LNG plant where it typically accounts for 30-40% of the capital cost of the overall plant. Liquefaction of natural gas involves the transfer of energy from hot stream of natural gas to cold stream of the refrigerant via heat exchangers. During this process, the phase of natural gas changes to liquid. The heat exchanger is a very important component in this heat exchange process. However, as time passes the performance of the heat exchanger decreases. These are due to fouling.

Fouling of heat exchangers is defined as the accumulation of unwanted deposits on heat transfer surfaces. The unwanted deposits layer imposes an additional resistance to heat transfer and the narrowing of the flow area. These unwanted deposits are usually hydro dynamically rough so that there will be an increased resistance to the flow of the fluid across the deposit surface [2], [3] This will reduce the efficiency of the heat exchanger and will cause other associated operating problems such as excessive pressure drop across the exchange.

Fouling cannot be avoided, but can be reduced. Generally, there are three ways to reduce fouling in heat exchangers, namely: a more efficient heat exchanger design, addition of anti-unwanted deposits chemicals and periodic cleaning of heat exchangers [4]. A more efficient heat exchanger design can reduce fouling in the heat exchanger. However it requires additional costs for the design and expansion of the operating area of the heat exchanger [5]. The addition of anti-unwanted deposit chemicals can reduce fouling up to 50% but the cost for these chemicals are very expensive and will contribute in higher production cost.

The other method is to conduct periodic cleaning of heat exchangers. Periodic cleaning of heat exchangers can improve and restore the efficiency of the heat exchangers. However it should be noted that cleaning will also contribute in increasing cost. So the cleaning schedule needs to be optimized to ensure that it is effective.

In optimizing heat exchanger, we need three components, namely the heat exchanger (HE) model, the fouling model and efficient optimization techniques. HE model is developed by assembling heat exchanger models [6].

For fouling model, empirical and theoretical models have been proposed to describe fouling in heat exchangers. Empirical models are generally used for the purpose of optimizing the schedule for clearing heat exchangers. In this project, the empirical fouling model used is linear model [7]. This is because at the time of cleaning, all systems will be shutdown. The cleaning scheduling optimization contains two types of variables, namely continuous and discrete variables in their objective functions [4]. This type of optimization is considered as Mixed Integer Nonlinear Programming (MINLP).

The complexity of the MINLP will depend on the objective function. In optimization of cleaning schedule, we need to consider several objective attributes such as to maximize energy output and to minimize cost. This function will produce non-convex output. Thus, it will not achieve optimal global solution. The MINLP problem belongs to a class of theoretically difficult problems and are very difficult to solve [8]. To solve MINLP problem, there are two approaches that can be used, namely deterministic approach and stochastic approach [9]. Deterministic approach is quite complicated as heat exchangers are connected and bounded with one another. In addition, the deterministic approach produces optimal local solutions [10]. Therefore in this paper, we choose to use a stochastic based approach.

A stochastic approach could give an optimal global solution. Previous researchers used different algorithms such as genetic algorithms (GA), particle swarm optimization (PSO) and independent component analysis (ICA) [11]. In this study, we develop an optimization for heat exchanger cleaning schedule using duelist algorithm (DA). Previous reports show that DA has a more optimal solution compared to using GA, PSO and ICA methods [12].

2 Approach

In this paper, we develop an optimization algorithm for the cleaning schedule of the heat exchanger. To achieve this objective, the work is separated into several steps. The first step is to develop and build a plant liquefaction simulation model. The second step is to find fouling characteristics from the plant model and develop a fouling model. The third step is to find and run operating conditions of the plant which can be used in solving the optimized cleaning scheduling. The four and last step is to develop the stochastic approach on existing data.

2.1 Plant Model

For the model, we considered a plant which has LNG cooling system in its heat exchanger. The coolant used is propane and Mixed Component Refrigerant (MCR) which is a mixture of nitrogen, methane, ethane and propane. Propane and MCR are obtained as the byproduct of LNG processing at the fractionation plant. The plant has 10 heat exchangers for the cooling process, 4 condensers for cooling, and 1 Main Cryogenic Heat Exchanger (MCHE).

In developing the model, data was obtained from one of the companies that processing natural gas into LNG. The collected data is the process flow diagram (PFD) and the distributed control system of liquefaction process. The PFD contains heat and mass balance data such as pressure, temperature, fluid flow rate, composition of the MCR, etc. DCS data is used to see the real condition of the process. There are some parameters that are not in either PFD or DCS data, so we use the data from the PFD model in the HYSYS software. The PFD of the liquefaction process used can be seen in Figure 1. The condition of natural gas feeds is shown in Table 1.

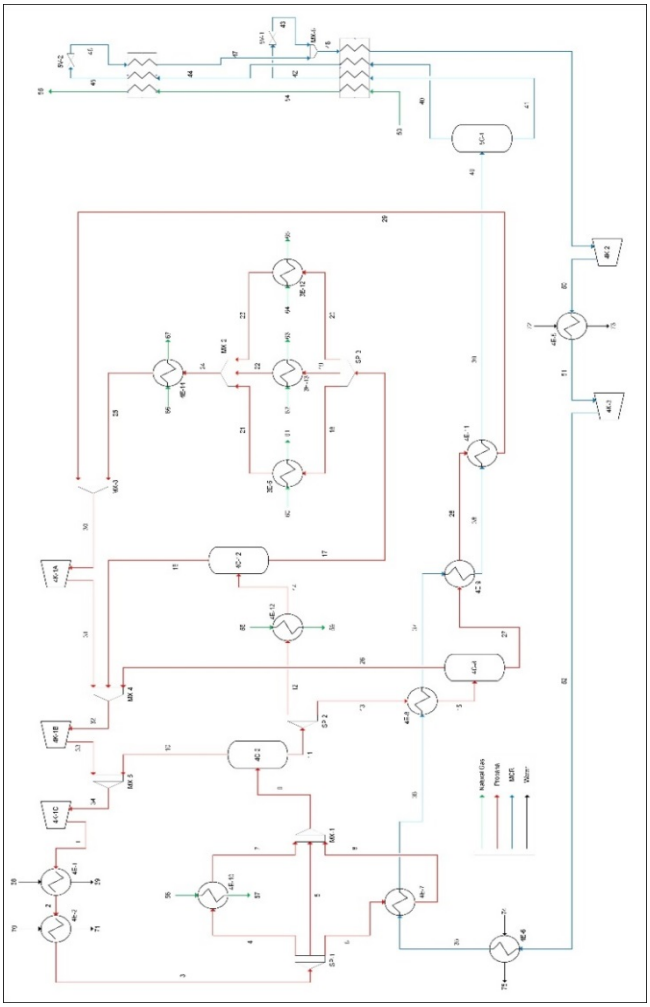


Figure 1 Process Flow Diagram

Table 1 Natural Gas Data used in the model

Variable	Unit	Value
Temperature	C	-35.340
Pressure	kPa	4216.860
Flowrate	kg/s	107.199
Composition		
Methane	mol%	0.9117
Ethane	mol%	0.0551
Propane	mol%	0.0272
i-Butane	mol%	0.0029
n-Butane	mol%	0.0021
i-Pentane	mol%	0.0001
Nitrogen	mol%	0.0009

2.2 Determination of Objective Function in Optimization

Regular heat exchanger cleaning can reduce fouling. However it should be noted that cleaning will impact the production and its costs. So that optimization of the cleaning schedule needs to be conducted. Optimization of the heat exchanger cleaning schedule is expressed in terms of the amount of cleaning within one plant operating period. The general objective of the optimization is to save maintenance costs and use energy. The formulation of scheduling optimization using mixed-integer-nonlinear-programing (MINLP) which is linearized using mixed-integer-linear-programing (MILP) to get the optimum global solution has been studied [13].

In this research, the objective function aims to get the maximum energy saving. The formulation of the problem used in optimizing the cleaning schedule in this study is as follows [14];

$$\max_{N_{c,n}} J = C_{\mathrm{E}} \sum_{n=1}^{N_{\mathrm{E}}} \sum_{t=1}^{N_{\mathrm{tf}}} \left(Q_{n,t}^{cs} - Q_{n,t}^{fc}\right) y_{n,t} - \sum_{n=1}^{N_{\mathrm{E}}} \sum_{J=1}^{N_{\mathrm{c,n}}} C_{\mathrm{Cl},n,j} (1 - y_{n,t}) - C_{\mathrm{p}} \sum_{n=1}^{N_{\mathrm{E}}} \sum_{t=1}^{N_{\mathrm{tf}}} \left(W_{Pn,t}\right) \quad (1)$$

C_E = Energy cost
C_{Cl} = Cleaning fee
C_p = Cost of working pumps
N_E = No of *heat exchanger*
$N_{c,\,n}$ = Number of heat exchanger cleanings for n years in the given operating period
N_{tf} = Number of days
$Q_{n,t}^{cl}$ = Heat transfer in *n* when *t* is clean
$Q_{n,t}^{fc}$ = Heat transfer in *n* when *t* is dirty
$Q_{n,t}^{cs}$ = Heat transfer in *n* at *t* with the cleaning schedule
W_p = Increased pump performance due to increased pressure drop
Y = Status of each heat exchanger, 1 if the unit is working and 0 if the unit is not working
n = 1,2 … N_E nomor *heat exchanger*

2.3 Optimization Using Duelist Algorithm

Duelist Algorithm is an evolutionary optimization based on genetic algorithms which is inspired by battle competition [12]. In the Duelist algorithm (DA), all individuals in the population are referred to as Duelists.

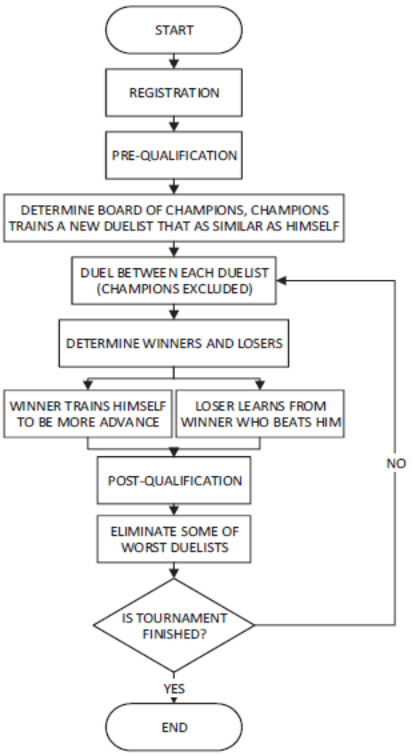


Figure 2 Flowchart of Duelist Algorithm

The flowchart of DA is shown in Figure 2. The cleaning intervals are represented as Duelists. The cleaning intervals are designed between 0 to 60 months. The optimized values are obtained by simulating heat exchanger and fouling models at different cleaning intervals during the time frame (6 years).

Steps for DA are as follows; 1) Registration of duelist candidates. Every duelist is set up by a registered duelist using various binary arrays. The arrays is called a skillset. 2) Pre-qualification. Pre-qualification is a test given to each duelists to measure or evaluate their fighting abilities based on their skillset. 3) Determination of the champion. The champion is determined to keep the best Duelist in the game. Every champion must train a new duelist to be like him. This new duelists will replace the champion position in the game and join the next duel. 4) Determine the duel schedule between each duelist. The duel schedule between each duelist is arranged randomly. Every duelist will fight using their abilities in duel and luck to determine the winner and loser. The duel uses simple logic. If duelist A is fighting ability plus his luck is higher than duelist B, then Duelist A is the winner and vice versa. The duelist's luck was determined by pure randomly to avoid the local optimum. 5) Improvement of duelist. After match, each duelist is categorized as a champion, winner and loser. To improve the abilities of each duelist there are three types of treatments for each category. For losers, every loser is improved by copying part of a winning skillset from the winner. For winners, each winner will improve their own abilities by random manipulations. And finally, each champion will train a new duelist. 6) Elimination. Because there are some duelists who have just joined the game, there must be eliminations to keep the duelists so that the quantity is the same as defined. This deletion is based on each duelist regarding his duel abilities. The duelist with the worst dueling ability is eliminated. It may be repeated until the maximum iteration is reached.

3 Result and Analysis Approach

3.1 Plant Modeling Validation

To validate the model, the flow rate of each process of the input for each component will be the same with the flow rate at the component output, assuming that there is no increase or decrease in mass in the process.

The results of the comparison can be seen in Table 2. From the table it is shown that the error was less than 10% for all cases indicating a good model result

Table 2 Plant Model Validation

Output of heat exchanger	Data		Model		Error	
	Tube (°K)	Shell (°K)	Tube (°K)	Shell (°K)	Tube (%)	Shell (%)
3E - 05	266.74	235.40	267.17	245.26	0.161	4.187
3E - 12	266.75	235.40	267.31	236.06	0.209	0.278
3E - 13	266.75	235.40	267.33	239.54	0.218	1.758
4E - 01	312.97	340.25	293.21	312.27	6.313	8.223
4E - 02	293.63	303.17	293.21	300.56	0.144	0.860
4E - 05	305.85	294.99	306.38	296.58	0.173	0.540
4E - 06	305.55	301.78	308.43	309.82	0.943	2.664
4E - 07	293.65	290.02	304.47	305.55	3.684	5.356
4E - 08	290.05	266.74	292.17	291.58	0.732	9.310
4E - 09	266.75	235.40	276.11	251.21	3.508	6.714
4E - 10	293.65	290.39	308.45	310.95	5.039	7.079
4E - 11	236.85	243.88	253.32	266.75	6.953	9.377
4E - 12	290.89	266.74	293.08	291.08	0.752	9.124
4E - 14	235.45	235.02	247.28	258.08	5.026	9.812

4.5 Optimization

The parameters used in optimization are: 1) Mechanical geometry of heat exchanger based on design data. 2) The inlet conditions of each heat exchanger. 3) Fouling resistance (Calculation of fouling resistance based on preliminary data). 4) The cleaning fee for each heat exchanger is set to $ 500,000. 5) The energy cost is set to 3.08 USD / GW. 6) Electricity tariff is 0.064 USD / kWh.

In Eq. 1, the objective function is affected by heat duty when it is dirty and there is fouling pressure drop caused by fouling reactivity and the cost of cleaning the heat exchanger. With optimization, fouling resistance improvements occur as shown in Figure 3. After cleaning on day 446 or the 15th month there is a decrease in fouling resistance at point 0 because after cleaning the fouling resistance disappears so that the heat exchanger is clean. This also applies on the 897th day or 30th month. When cleaning the heat exchanger takes place, the time required for cleaning is 5 days calculated from the shutdown when cleaning the heat exchanger.

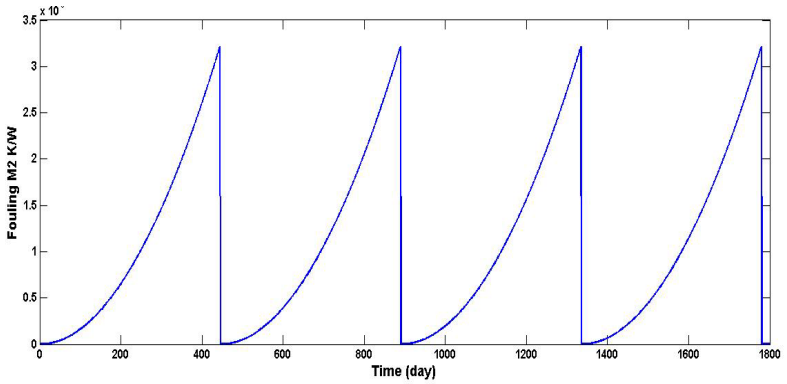


Figure 3 Fouling resistance after optimization

In the 15th month to the 30th month the heat exchanger has the same trend as the 0 to 15 months. This is because in the 15th month the heat exchanger undergoes cleaning so the fouling resistance trend returns to normal. In the 30th month it also had the same trend as the previous trend.

One of the driving force for the optimization is to save economically. Using the predefined data, we can get the cost before and after optimization. Table 3 shows the cost of one heat exchanger before cleaning for one period (6 years).

Table 3 Cost before Cleaning

Heat Exchanger (HE)	Clean HE			With Fouling HE		
	Recovery Energy (USD)	Pump Cost (USD)	Total Recovery Energy (USD)	Recovery Energy (USD)	Pump Cost (USD)	Total Recovery Energy (USD)
1	5,410,043	39,724	5,370,319	3,993,067	49,748	3,944,319

Table 4 shows the cost of the heat exchanger after implementing optimized schedule for cleaning.

Table 4 Cost after Cleaning

Heat Exchanger	Optimized Cleaning Schedule			
	Recovery Energy (USD)	Cleaning Cost (USD)	Pump Cost (USD)	Total Recovery Energy (USD)
1	4,881,913	54,000	42,163	4,785,150

We can see that we can save from the total energy recovery heat exchanger in conditions after optimization (Energy recovery - cleaning costs - pump costs) reduced by the total energy recovery heat exchanger in gross conditions (energy recovery with fouling - pump costs with fouling). The total energy recovery heat exchanger after optimization is $ 3,944 million minus the total energy recovery heat exchanger in gross conditions of $ 4,785 million, a profit of $ 0.841 million or equivalent to $ 841,000.

Table 5 shows the optimization results.

Table 5 Optimization Results

Variabel	Unit	With Fouling	Optimized	Recovery Energy
Heat Duty	MJ/s	138,737 MJ/s	585,001 MJ/s	446,264 MJ/s
Pump Power	MWh	895,446MWh	213,527MWh	681,919 MWh
Saving	USD ($)	$3.944 millions	$4.785 millions	$841.000

It can be seen that when the heat exchanger is cleaned, we can save 446,264 MJ/s (heat duty), 681,919 MWh (pumps power) and $ 841,000.

4 Summary

The need for energy is increasing worldwide including the need of natural gas. LNG is natural gas that has been cooled to –260° F (–162° C), changing it from a gas into a liquid that is 1/600th of its original volume. Liquefaction of natural gas involves the transfer of energy from hot stream of natural gas to cold stream of the refrigerant via heat exchangers. However, as time passes the performance of the heat exchanger decreases due to fouling.

Fouling of heat exchangers cannot be avoided, but can be reduced by conducting periodic cleaning of heat exchangers. Periodic cleaning of heat exchangers can improve and restore the efficiency of the heat exchangers. However it should be noted that cleaning will also contribute in increasing cost. In this paper, we develop an optimization algorithm for the cleaning schedule of the heat exchanger.

First we develop plant model based on actual plant. Next, fouling characteristics are defined based on analytical approach and included in objective function equation which is used in solving the optimized cleaning scheduling. The optimization approach is based on stochastic approach knows as Duelist Algorithm. Results find that the proposed method is able to save 446,264 MJ/s (heat duty), 681,919 MWh (pumps power) and $ 841,000 in the period 72 months.

Acknowledgements We would like to thank Department of Electrical and Electronic Engineering, University of Nottingham Malaysia and Teknik Fisika, Institut Teknologi Sepuluh Nopember for partially funding this project.

References

[1] BP, “BP Energy Outlook 2035,” *Energy Outlook 2035*, 2014.
[2] N. Epstein, *Fouling of Heat Exchangers*. 1986.
[3] H. Müller-Steinhagen, M. R. Malayeri, and A. P. Watkinson, “Heat exchanger fouling: Mitigation and cleaning strategies,” in *Heat Transfer Engineering*, 2011.
[4] F. Smaïli, V. S. Vassiliadis, and D. I. Wilson, “Mitigation of fouling in refinery heat exchanger networks by optimal management of cleaning,” *Energy and Fuels*, 2001.
[5] A. C. Caputo, P. M. Pelagagge, and P. Salini, “Joint economic optimization of heat exchanger design and maintenance policy,” *Appl. Therm. Eng.*, 2011.
[6] M. Markowski and K. Urbaniec, “Optimal cleaning schedule for heat exchangers in a heat exchanger network,” in *Applied Thermal Engineering*, 2005.
[7] E. Andersson, J. Quah, and G. Polley, “Experience in the Application of Compabloc TM in Refinery Pre-Heat Trains and First Analysis of Data From an Operational Unit,” *Int. Conf. Heat Exch. Fouling Clean. VIII*, 2009.
[8] D. R. Woods, S. J. Anderson, and S. L. Norman, “Evaluation of capital cost data: Heat exchangers,” *The Canadian Journal of Chemical Engineering*. 1976.
[9] M. Errico, S. Maccioni, G. Tola, and P. Zuddas, “A deterministic algorithm for the synthesis of maximum energy recovery heat exchanger network,” *Comput. Chem. Eng.*, 2007.
[10] A. L. Diaby, S. J. Miklavcic, and J. Addai-Mensah, “Optimization of scheduled cleaning of fouled heat exchanger network under ageing using genetic algorithm,” *Chem. Eng. Res. Des.*, 2016.
[11] J. M. Zamora and I. E. Grossmann, “A global MINLP optimization algorithm for the synthesis of heat exchanger networks with no stream splits,” *Comput. Chem. Eng.*, 1998.

[12] T. R. Biyanto *et al.*, "Duelist algorithm: An algorithm inspired by how duelist improve their capabilities in a duel," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.
[13] M. C. Georgiadis, L. G. Papageorgiou, and S. Macchietto, "Optimal cleaning policies in heat exchanger networks under rapid fouling," *Ind. Eng. Chem. Res.*, 2000.
[14] T. R. Biyanto, M. Ramasamy, and H. Zabiri, "Modeling heat exchanger using neural networks," *2007 Int. Conf. Intell. Adv. Syst.*, 2007.

MISTA 2019

Integrating load based controlled release with priority dispatching for dynamic flexible assembly job shop with alternative process plans

Hongyang Zhong • Jianjun Liu • QingXin Chen • Zhantao Li

Abstract Scheduling of the dynamic flexible assembly job shop scheduling problem (DFAJSP) is of great importance in fields of production management, while it is extremely hard to find an optimal solution. This paper deals with DFAJSP considering three phase of control: load based controlled release, process planning and priority dispatching. Priority dispatching can react in time to unexpected events, but it only makes decisions locally. Thus, load based controlled release is introduced to make use of global information and balance the workloads. Besides, a new work route decision policy is developed to improve the priority rule, which determines an optimal process routes for job to be released and in process. The aim of this paper is to evaluate the performance of different combinations of the priority rules under load based controlled release methods. Full factorial simulation experiment has been designed for the DFAJSP, results shows that route flexibility is highly relevant to the due date and lead time performance, and the proposed work route decision policy cooperate better with the load based controlled release in lead time based performance compared with other benchmarks.

Key Words: Flexible assembly job shop; Controlled release; Process planning; Priority dispatching; Simulation

1 Introduction

Hongyang Zhong
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: windrose@foxmail.com

Jianjun Liu
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: jianjun33@163.com

Qingxin Chen
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: qxchen@gdut.edu.cn

Zhantao Li
Guangdong University of Technology, Key Laboratory of Computer Integrated Manufacturing System of Guangdong Province
E-mail: lzt0@163.com

Job shop scheduling (JSP), one of a typical strong NP-hard combinational optimization problems, has received noteworthy attention from researches and practitioners over the last few decades. In classical JSP, there are two basic assumpations, which are: i) each job has only one process plan, and ii) jobs are independent of each other. With the development of flexible manufacturing system and Computer Aided Process Planning (CAPP), a given operation can be performed by more than one machines, and a job can be processed by different combination of operations (e.g. Tan & Khoshnevis 2000; Li et al 2010; Phanden, Rakesh et al 2011). The first restriction of JSP is relaxed and JSP extends to Flexible job shop scheduling (FJSP), and to relax the second restriction, jobs are related and after their completions, they are assembled into a product according to the Bill-Of-Material (BOM). In this way, FJSP develops into the Flexible assembly job shop scheduling (FAJSP). FAJSP consists of two sub-problems, which are: i) how to select process plan for a job, and ii) how to schedule process sequence of machines so as to coordinate the progress of related parts. In real production, a previously optimal schedule may get poor system performance or even becomes infeasible due to unpredictable real time events (e.g. new order arrival), dynamic scheduling is of great importance to real production, thus this paper studies the dynamic flexible assembly job shop scheduling (DFAJSP) as research object.

Generally, there are mainly three categories of dynamic scheduling technologies (Ouelhadj et al 2008), which are completely reactive, predictive-reactive, and proactive reactive scheduling. In completely reactive scheduling, no firm schedule is generated and decisions are made in real time, so priority dispatching rules are frequently used to select the next processing job with highest priority once the machine become free. Priority dispatching rule is usually intuitive, and easy to implement, but it is also myopic and make real-time decision based on local information (e.g. Doh et al 2013; Thürer, et al 2012; Lu, et al 2011). Predictive-reactive scheduling is a scheduling/rescheduling process in which schedules are revised in response to real time events(Jin et al 2015), and pro-active scheduling focus on building predictive schedules in advance to satisfy performance in a dynamic environment(Yu et al 2006; Fattahi et al 2012). They both uses the global information and searches in a larger solution space compared with completely reactive scheduling, but they may fail to respond quickly and in time to real time events. In order to prevent the myopia of completely reactive scheduling, load based controlled release methods (e.g. Thürer, et al 2012) are introduced. An integrated approach (e.g. Lu, et al 2011) combining controlled release and priority dispatch is established to make use of global information and cope with the disturbance.

Previous studies show that combining route decision with load based order release is capable of balancing the workloads across workstations (e.g. Kingsman, et al 1996), but they all neglect the fact that a job has alternative routings and machines to select after release. Hence, the purpose of this paper aims at proposing a new routing decision policy to make full use of routing flexibility, and we apply it to the integrated approach that combines order release with process planning. A simulation study is conducted to evaluate the routing decision with proposed policy compared with that without that policy, finally the results are presented and discussed.

2 Problem description

In the DFAJSP considered in this study, there are n workstations in the shop floor, each workstation is able to perform different types of operations, they have equal capacities, the network model of the job's alternative process plans is presented in Fig.1, and this study is based on the following assumptions:

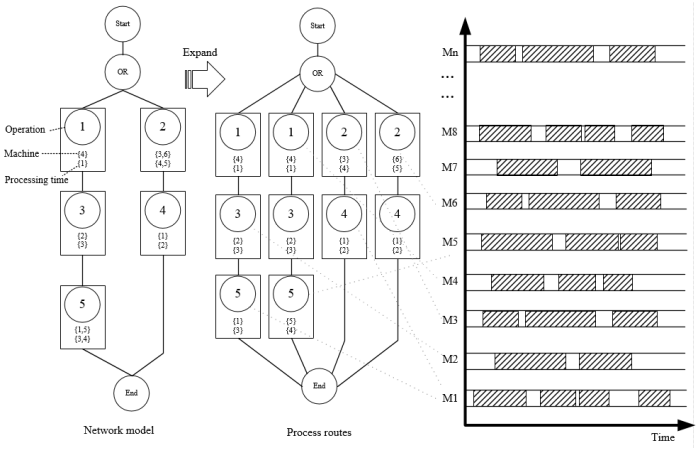


Fig.1

(1) order *i* arrive at the job pool stochastically, suppose each order only contain single product, which consist of several parts, the network model of part *j*'s process routes, alternative machines for all operations and their corresponding processing times are known after the order's arrival.
(2) Each workstation has the same number of machines with equal capacities, and this study assumes the workstation has only one machine with a process sequence. Each workstation can be visited at most once in the process routes of a job.
(3) Each machine can process one operation at a time, and pre-emption is not allowed, i.e. once a job is processed on a machine, it will stay on that machine until its completion.
(4) Set-up times for jobs are included in the processing times of jobs, and transportation times are negligible compared with processing times
(5) Assembly operation of the product is performed only if all of its belonging parts are completed.
(6) This paper focus on the manufacturing stage to make sure the completion of parts is close to the planned start time of assembly, and the assembly time is ignored in order to prevent the assembly operation becomes the bottleneck.
(7) Once the product is assembled, it can be delivered ahead of it due date.

2.1 Sets and indices

i: products ($i \in I$) ;	*j*: parts or subassemblies belonging to product i ,($j \in J_i$) ;
p: Index for process routes of job *j*， $p \in P_{ij}$, where P_{ij} denotes the set of routes for job *j* ;	O_{ijp}: Ordered set of operations in process route *p* of job *j*;
k: Operation k of process route *p* of job *j*, the first operation, where $k \in O_{ijp}$ (note that the first operation is denoted as k_f, while the last one is denoted as k_l);	*M*: Machines, $m \in M$
M_{ijpk} : Sets of alternative machines on which operation k ($k \in O_{ijp}$) can be processed, $M_{ijpk} \subseteq M$;	Pt_{ijpk}^{m} :Processing time of operation k on machine m;
At_i: assembly time of product *i*;	*L*: Large number;
R_i: arrival time of product *i*;	bu_m :the time that machine m will be idled in the start of the rescheduling period;
RT: start time of the rescheduling period;	

2.2 Decision variables

$w_{ijp}=1$ if process route p of job j is selected and 0 otherwise;

$x_{ijpk}^{m}=1$ if machine m is selected for operation k and 0 otherwise;

$y_{ijki'j'k'}^{m}=1$ if operation k of job j precedes operation k' of job j' on machine m and 0 otherwise;

S_{ijpk}^{m} : Start time of operation k on machine m;

C_{ijpk}^{m} : Completion time of operation k on machine m;

C_i : Completion time of product i;

2.3 Objectives

$$LT_i=\sum_{i\in I}(C_i-A_i)/|I| \tag{1}$$

$$C_i\geq\sum_{p\in P_{ij}}\sum_{m\in M_{ijpk'}}C_{ijpk'}^{m}+At_i\ ,i\in I,j\in J_i \tag{2}$$

$$\sum_{p\in P_{ij}}w_{ijp}=1,i\in I,j\in J_i \tag{3}$$

$$\sum_{p\in P_{ij}}\sum_{m\in M_{ijpk}}x_{ijpk}^{m}=1,i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp} \tag{4}$$

$$S_{ijpk}^{m}+C_{ijpk}^{m}\leq L\cdot x_{ijpk}^{m}\ ,i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp},m\in M_{ijpk} \tag{5}$$

$$\begin{gathered}C_{ijpk}^{m}\geq S_{ijpk}^{m}+Pt_{ijpk}^{m}-L\cdot(1-x_{ijpk}^{m}),\\ i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp},m\in M_{ijpk}\end{gathered} \tag{6}$$

$$\begin{gathered}\sum_{m\in M_{ijpk}}S_{ijpk}^{m}\geq\sum_{m\in M_{ijpk}}x_{ijpk}^{m}\cdot bu_m+RT\ ,\\ i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp},m\in M_{ijpk}\end{gathered} \tag{7}$$

$$\sum_{m\in M_{ijpk}}S_{ijpk}^{m}\leq\sum_{m\in M_{ijpk}}C_{ijp(k-1)}^{m}\ ,i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp}\left\{k_f\right\} \tag{8}$$

$$\begin{gathered}\sum_{p\in P_{ij}}S_{ijpk}^{m}\geq\sum_{p\in P_{i'j'}}C_{i'j'pk'}^{m}-L\cdot y_{ijki'j'k'}^{m},\\ i,i'\in I,j\in J_i,j'\in J_{i'},k\in O_{ijp},k'\in O_{i'j'p},m\in M_{ijpk}\bigcap M_{i'j'pk'}\end{gathered} \tag{9}$$

$$\begin{gathered}\sum_{p\in P_{i'j'}}S_{i'j'pk'}^{m}\geq\sum_{p\in P_{ij}}C_{ijpk}^{m}-L\cdot(1-y_{ijki'j'k'}^{m}),\\ i,i'\in I,j\in J_i,j'\in J_{i'},k\in O_{ijp},k'\in O_{i'j'p},m\in M_{ijpk}\bigcap M_{i'j'pk'}\end{gathered} \tag{10}$$

$$S_{ijpk}^{m},C_{ijpk}^{m},C_i\geq 0\ ,i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp},m\in M_{ijpk} \tag{11}$$

$$w_{ijp},x_{ijpk}^{m}\in\left\{0,1\right\},i\in I,j\in J_i,p\in P_{ij},k\in O_{ijp},m\in M_{ijpk} \tag{12}$$

$$y^{m}_{ijki'j'k'} \in \{0,1\},$$
$$i,i' \in I, j \in J_i, j' \in J_{i'}, k \in O_{ijp}, k' \in O_{i'j'p}, m \in M_{ijpk} \bigcap M_{i'j'pk'} \quad (13)$$

Formulation (1) is objective function to minimize the mean product lead time, where $|I|$ is the cardinality of set I, constraint (2) defines the completion time of a product, i.e. the completion time of latest finished part belonging to the product plus the assembly time. Constraint (3) represents that only one process route must be chosen for each job. Constraint (4) guarantees that each operation of a job is assigned to only one machine under the circumstance that the operation within the process route is selected. Constraint (5) and (6) specifies the start and completion time of each operation. Constraint (7) ensures the start time of operation must be greater than the idle time of its alternative machines. Constraint (8) specifies the precedence relationships between operations for a process route of a job are not violated. Constraint (9) and (10) make sure that two operations in their selected process routes are not processed simultaneously on any machine in the joint set of their alternative machines. Last but not least, the other constraints (11)、(12)、(13) represent the conditions on the decision variables.

Though the mixed linear programming model can be solved by commercial software package, it consumes excessive computation time when the size of the instance gets large. We used Gurobi 8.1.1 solver to testify the validation of the mathematical model, the Gurobi solver could work well until the instance size goes up to 20 jobs and 12 machines. Hence, we suggest to propose priority rules to solve the problem instead of developing new optimal algorithms, besides, we also integrate load-based order release methods with the priority rules to overcome the weakness that priority rules only make decisions locally in real-time (i.e. select the next processing job based on the local information in queue when the machine become free).

3 An integrated approach for solving DFAJSP

The integrated approach uses a combination of release methods and priority rules to solve DFAJSP. Since the DFAJSP consists of three sub-problems, Load based controlled release methods and two types of priority rules are needed, i.e. one for process routes selection and other for job shop scheduling. In this study, we test 12 integrated approaches, i.e. 12 combinations of four route sequencing rules and three priority dispatching rules. These rules were chosen because they represent a certain type of attribute belonging to the jobs. Before describing the priority rules, the additional notations used are summarized below.

DD: due date of product i ($i \in I$) ;	LC_m: Aggregate load of workstation m
WL_m: workload norm of workstation m	RL_p: the number of operations in process routes p
d: estimated throughput time of a workstation, constant	UO_{ijp}: the number of unfinished operations in process routes p

3.1 Refined load-based order release methods for DFAJSP

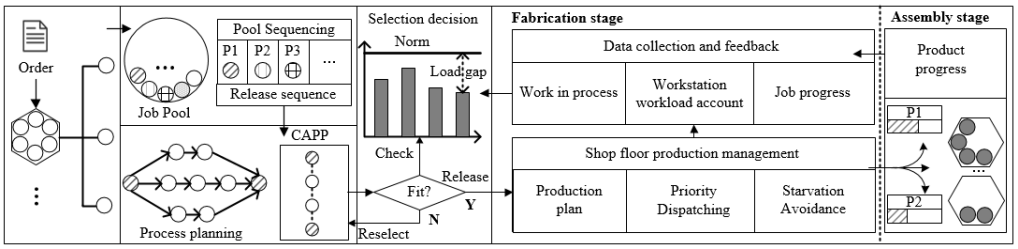


Fig.2

Fig.2 presents the coupled controlled release method applied to DFAJSP. Our literature review showed that LUMS COR (Lancaster University Management School Corrected Order

Release) has been identified as the best performing load-based order release method for assembly shops (Thürer, et al 2012). However, LUMS COR has not ever been applied in flexible assembly job shop, so we first refine LUMS COR by considering routing flexibility in both periodic release and starvation avoidance mechanism:

Decision	Details

Periodic release control	Triggering: at fixed time interval
	All jobs in the pool are sequenced according to their PRDs which is obtained by backward scheduling
	For j in J do:
	All process plans are sequenced according to a certain attribute
	For p in P do:
	If $\frac{PT_{ijpkm}}{k} + LC_m \le WL_m$ Then
	Job j is released and removed from J, the load is updated $LC_m := LC_m + \frac{PT_{ijpkm}}{k}$
	Break
	else
	The next routing is checked until all routes are checked.
	End;
	If no routes can fit the norms, Job j remains in the pool until next release decision
	End;

Starvation avoidance mechanism	Triggering: Any workstation is starving, i.e. machine is not occupied and the queue has no job All jobs in the job pool are sequenced according to their PRDs For j in J do: All process plans are sequenced according to a certain attribute For p in P do: If First operation of p is the idle workstation then: Job j is released and removed from J, the load is updated $LC_m := LC_m + \frac{PT_{ijpkm}}{k}$ Break; End; Break; End;

3.2 Work route decision policy

Once the alternative machine is determined for each operation of a job, the process route is determined. Our literature review (e.g. Henrich, et al 2006; Kingsman, et al 1996) shows that there are five stages for route decision, and route decision during order release stage may lead to a well-balanced workload. However, the higher ranked (by route sequencing rule) route cannot always be selected because of the upper bound constraints, and a lower ranked route may often fit in the norm. Previous studies assume the route of a job is fixed until its completion, but here we propose a new work route decision policy that predetermine a route at the order release stage, and allow a job to reselect alternative machine when it is going to the next operation. The procedure of the changeable work route decision policy is presented in Fig.3:

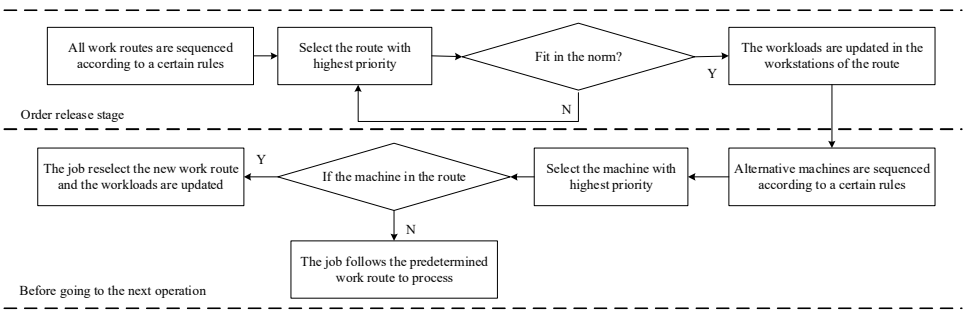


Fig.3

3.3 Route sequencing rules

RS (Random selection): select a process route randomly from its alternative process routes.
RSP (Route shortest processing time first): select a process route with the shortest aggregate processing time within its alternative process routes. i.e. select a process route p that satisfies the following condition.

$$p^* = \arg\min_{p \in P_{ij}} \{ \sum_{k \in O_{ijp}} Pt_{ijpkm} \}$$

RSW (Route smallest mean workload first): select a process route with the smallest aggregate. i.e. select a process route p that satisfies the following condition.

$$p^* = \arg\min_{p \in P_{ij}} \left\{ \sum_{k \in O_{ijp}} (LC_m) / RL_{ijp} \right\}$$

MRSW (Modified route smallest mean workload first): A new route sequencing rule which is based on the work route decision policy proposed in Section 3.4. In order release stage, the process route with the smallest mean workload is selected, and whenever an operation is completed, all alternative machines are reselected according to the RSW rule.

3.4 Priority dispatching rules

FCFS (First come first serve): select a job with the earliest entry time at the workstation.
PST (Planned start time): select a job with the earliest planned start time at this operation. i.e. select a job j that satisfies the following condition.

$$j^* = \arg\min_{j \in Q_m} \{ DD_i - d \cdot UO_{ijp} \}$$

TWKR (Total work content remaining): select a job belonging to the product with the smallest total work content remaining, if two jobs belonging to the same product are in the same queue, IR (importance ratio) is used as the tie breaker.

$$j^* = \arg\min_{j \in Q_m} \{ \sum_{j \in J_i} \sum_{k \in O_{ijp}} Pt_{ijpkm} \}$$

The simulation model was developed using Plant Simulation 11.3 R software. A flexible job shop configuration with 12 workstations has been considered. Our purpose is to investigate the cooperative behaviors of the refined controlled release methods with dispatching rules. Table 1 summarizes the characteristics of the simulation model.

Table 1. Job and shop characteristics.

Routing variability	Random routing; no re-entrant flows
No. of workstations	12
Workstations capacities	All equal and constant over time
Workstation utilization rate	90%
No. of parts per product	Discrete Uniform [2,6]
No. of process plan	Low: 1; Medium: 2; High:3
No. of alternative machines per operation	Discrete Uniform [1,2]
No. of operations per job	Discrete Uniform [4,6]
Operation processing times	U~[0.7,1.8]
Due dates	Due date= Entry Time+d; d U~[40,70]
Inter-Arrival Times	Exp. Distribution (mean set to achieve 90% utilization)

The experimental factors and the levels at which they are evaluated are summarized in Table 2. These are: (i) four route sequencing rules (RS, RSP, RSW, MRSW); (ii) three priority

dispatching rules (FCFS, PST, TWKR); (iii) six workload norms (6.3, 8.5,10.8,13.1,15.3,17,5); (iv) three level of route flexibility, i.e. the number of alternative process routes.

Table 2. Experimental Factors and levels.

Factors	Levels
Route Sequencing rule	RS RSP RSW MRSW
Dispatching rule	FCFS PST TWKR
No. of process plans	Low:1 Moderate:2 High:3
Workload norm levels	6.3 8.5 10.8 13.1 15.3 17.5

Four main performance measures are considered in this study: which are classified into lead-time based and due-date based performance measures. The former includes product lead time (PLT) and shop floor throughput time, while the later includes the percentage of tardy products (PTP) and product tardiness (PT), these measures are defined in Table 3.

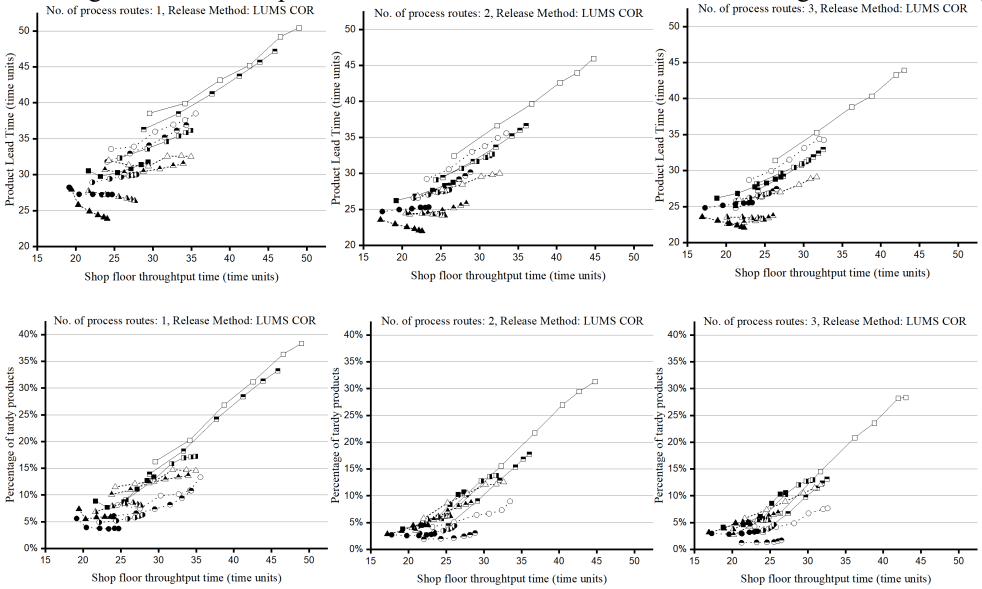
Table 3. Summary of performance measures

Name	Brief description
Product Lead Time (PLT)	completion time of the product (all parts belonging to the same product are completed) minus the product arriving time
Shop Floor Throughput Time (SFTT)	completion time of the product minus the part's release time
Product Tardiness (PT)	The conditional lateness, i.e. max (0, C-D), product completion time minus its due date
Percentage of Tardy Products (PTP)	The percentage of products that are completed after its due date

4 Results and discussion

The results of the simulation experiments are presented in Fig.4, the x-axis is the shop floor throughput time (SFTT), which represents the workload norm level (increase stepwise from the lowest to the highest as the x get larger), and the y-axis is the other three performance measures defined previously.

Three figure in a row represents three levels of routing flexibility, the left figure stands for the lowest routing flexibility, and the middle stands for the moderate setting, and the right one for the highest. As routing flexibility gets higher in a tight workload norm, the performance of shop floor throughput time (SFTT) gets better, high flexibility may compensate for the incoordination caused by tight norm. As the route gets less flexible and the norm become looser, the SFTT gets worse. Other performance measures can also be enhanced in high route flexibility.



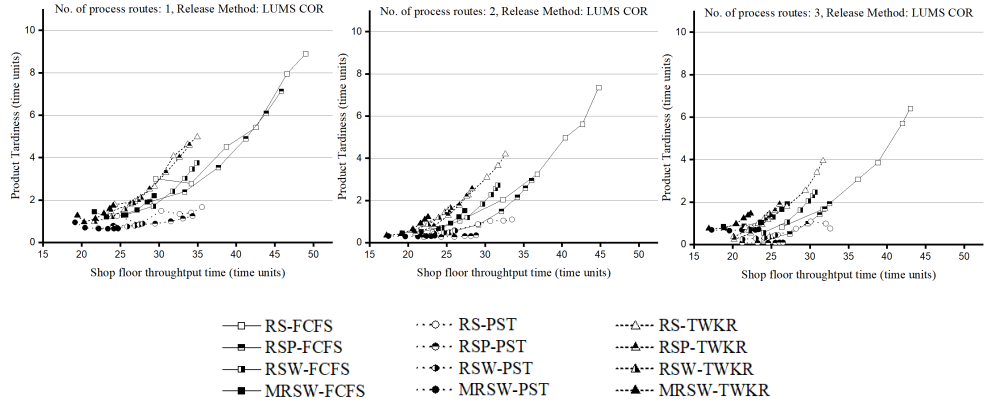


Fig.4

The most general observation is that MRSW rule based on the new proposed work route decision policy can obtain the best performance in lead time based measures (e.g. PLT and SFTT) at all routing flexibility levels. As for the due date based measures (PTP and PT), MRSW rule can obtain a better performance when the route flexibility is relatively low.

The performance of RS route sequencing rule is quite sensitive to the norm, while RSW, RSP, MRSW is much more stable. MRSW rule, which is on the basis of RSW rule, totally outperform RSW. RSW rule can obtain a better performance in all performance than RSP rule when the routing flexibility is low, but as the routing flexibility gets high, the RSP rule outperform RSW rule and even MRSW rule.

The performance of TWKR priority dispatching rule outperforms any other rule in Product lead time (PLT), but TWKR rule is much sensitive to the change of workload norm as it gets worse more quickly when norm gets looser. PST rule gets the best performance in due date based performance measures (PT and PTP).

As for the cooperative function of route sequencing rule and priority dispatching rule, the integrated approach can combine the strengths in both route sequencing rules and priority dispatching rules. MRSW-TWKR rule gets the best performance in PLT in all routing flexibility. When alternative routes is few, the MRSW-PST rule can get better performance in due date performance like Product tardiness PT and PTP, but as the alternative process plan gets more, the RSP-PST rule is superior to other integrated rules in due date based measures.

5 Conclusion

The overall research perspective was to investigate the cooperative function of routing sequencing rule and priority dispatching rule within the effective use of load-based order release methods in flexible assembly job shop. Whereas prior studies mainly focus on work route decision in order release stage in flexible job shop, we propose a new work route decision policy by reselect machine after job release, and MRSW rule is developed within the policy.

The simulation results shows that reselecting machine after job release can be beneficial in shorten product lead time compared with a fixed routing policy, besides, the policy can improve delivery reliability when routing flexibility is low. RSW rule based on workload can outperform RSP rule when alternative process plan is few, but RSP rule is superior to RSW rule when routing flexibility gets higher.

The coordination improved priority rule (e.g. TWKR rule) can obtain a better lead time performance, while the urgency based priority rule (e.g. PST rule) can achieve a better delivery reliability. The integrated approach can combine the strengths of route sequencing rules and priority rules and no combined rules is superior to any other rules in all performance measures.

Acknowledgements:

the National Natural Science Foundation of China (71572049,51975129,61573109), the S pecial Plan Young Topnotch Talent of Guangdong (2016TQ03X364), and the Pearl River S&T Nova Program of Guangzhou (201710010004)

References

1. Tan, Wei , and B. Khoshnevis . "Integration of process planning and scheduling— a review." Journal of Intelligent Manufacturing 11.1(2000):51-63.
2. Li, Xinyu , et al. "A review on Integrated Process Planning and Scheduling." International Journal of Manufacturing Research 5.2(2010):161-180.
3. Phanden, Rakesh Kumar , A. Jain , and R. Verma . "Integration of process planning and scheduling: a state-of-the-art review." International Journal of Computer Integrated Manufacturing 24.6(2011):517-534.
4. Doh, Hyoung Ho, et al. "A priority scheduling approach for flexible job shops with multiple process plans." International Journal of Production Research51.12(2013):3748-3764.
5. Fernandes, Nuno O. , et al. "Improving workload control order release: Incorporating a starvation avoidance trigger into continuous release." International Journal of Production Economics (2016):S0925527316304091.
6. Matthias Thürer, et al. "Workload Control and Order Release: A Lean Solution for Make‐to order Companies." Production and Operations Management 21.5(2012).
7. Lu, H. L. , G. Q. Huang , and H. D. Yang . "Integrating order review/release and dispatching rules for assembly job shop scheduling using a simulation approach." International Journal of Production Research 49.3(2011):647-669.
8. Fattahi, Parviz , and A. Fallahi . "Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability." CIRP Journal of Manufacturing Science and Technology 2.2(2010):114-123.
9. Thürer, M, et al. "The application of workload control in assembly job shops: an assessment by simulation." International Journal of Production Research 50.18(2012):5048-5062.
10. Jin, Liangliang , C. Zhang , and X. Shao . "An effective hybrid honey bee mating optimization algorithm for integrated process planning and scheduling problems." The International Journal of Advanced Manufacturing Technology 80.5-8(2015):1253-1264.
11. Yu, Xuefeng , and B. Ram . "Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups." International Journal of Production Research 44.22(2006):4793-4813.
12. Ouelhadj, Djamila , and S. Petrovic . "A survey of dynamic scheduling in manufacturing systems." Journal of Scheduling 12.4(2009):417-431.
13. Henrich, Peter , M. Land , and G. Gaalman . "Grouping machines for effective workload control." International Journal of Production Economics 104.1(2006):125-142.
14. Kingsman, Brian , et al. "Responding to customer enquiries in make-to-order companies Problems and solutions." International Journal of Production Economics 46-47.none(1996):219-231.

MISTA 2019

A Multi-Commodity Unpaired Pickup and Delivery Vehicle Routing Problem with Split Loads and Unloads

Dongyang Xu• Kunpeng Li• Qingguo Bai• Ligang Cui

Abstract This paper addresses a multi-commodity unpaired pickup and delivery vehicle routing problem. In this problem, the pair of pickup and delivery among customers is unknown beforehand, and splits of pickup and delivery for a customer are allowed, respectively. Thus, both the pickup-delivery pairing decision and complicated vehicle routing decision are considered in the proposed problem, simultaneously. Firstly, an improved mixed integer programming model is built with seven types of constraints. Secondly, five types of valid inequalities for tightening the solution spaces are derived. Thirdly, an improved variable neighborhood search algorithm is presented combining the developed multi-start initial solution strategy and modified neighborhood local search procedure. Finally, extensive experiments are conducted on instances in previous literature to verify the performance of the proposed algorithm and indicate management insights. Experimental results demonstrate that: (1) with less decision variables and constraints considered, the proposed model adapts to more generalized scenarios and can solve more instances comparing to the existing models in previous literature. (2) The valid inequalities utilized to tighten the searching spaces can efficiently help the model obtain optimal solutions or high-quality lower bounds. (3) The improved algorithm is efficient to obtain optimal or near optimal solutions and superior to the comparing algorithms in terms of solution quality, computational time and robustness.

Keywords: Transportation; Vehicle Routing; Unpaired Pickup and Delivery; Split Loads and Unloads; Variable Neighborhood Search

Dongyang Xu
Institute of Modern Logistics, Henan University, Kaifeng 475004, China
School of Business, Henan University, Kaifeng 475005, China
E-mail: Second.xdy_hust @163.com

Kunpeng Li
School of Management, Huazhong University of Science & Technology, Wuhan 430074, China. E-mail: Second.likp @mail.hust.edu.cn

Qingguo Bai
School of Management, Qufu Normal University, Rizhao, 276826, China
E-mail: Second.qfnubaiqg @163.com

Ligang Cui
School of Economics and Management, Chongqing Jiaotong University, Chongqing, 400074, China. E-mail: Second.cligang @126.com

1 Introduction

Pickup and delivery vehicle routing problem (PDVRP) is a variant of the well-known vehicle routing problem (VRP). For PDVRP, a fleet of vehicles drives through the routes to satisfy a series of transportation requests. Each transportation request is associated with the load size and the location where it is picked up and delivered [1]. In general, PDVRP can be divided into two categories: paired and unpaired. Paired PDVRP (PPDVRP) indicates that each transportation request is uniquely specified by a pickup location, a delivery location, and a demand to be transported between these two locations [2]. Therefore, the pair of pickup and delivery should be predefined. For unpaired PDVRP, the pickup points and the corresponding pickup quantity (from each pickup point) for each delivery request are unknown beforehand [3-5]. Thus, the pair of pickup and delivery is a decision variable. UPDVRP is more complicated than PPDVRP. On one hand, UPDVRP needs to consider the pickup and delivery pairing decision when making the routing decision. On the other hand, these two decisions, pairing and routing, interact with each other (i.e., the change of pickup and delivery pairing decision will affect the corresponding optimal routing decision, and vice versa), which further increases the complexity of problem.

This paper explores an UPDVRP with multiple customers and multiple commodities, which has many applications in the real world. For example, in the retailing industry, newly designed commodities are commonly sold in different stores simultaneously. After a period of sailing, some commodities with specific colors or sizes are out of stock in certain stores, but commodities with the same colors or sizes are still in-stock at other stores. Generally, trucks are assigned to transfer commodities among stores (i.e., from sufficient stores to insufficient stores), in order to create more sales opportunities. Besides, similar application can be observed as that the bikes re-balance among stations in sharing systems, the raw materials (i.e., tobacco leaf) transferring among plants in multi-point production enterprises (i.e., tobacco industry). The above different application scenarios can be normalized as commodities transshipment among customers. Fig.1 presents an example illustrating the transshipment network with four customers and four commodities. In the network, the pickup customer and delivery customer is not separated, namely, each customer not only can be pickup customer for some commodities, but also can be delivery customer for other commodities. Furthermore, for a delivery customer of a commodity, in addition to the delivery routing, the pickup customers and the corresponding pickup quantity from every pickup customer need to be considered.

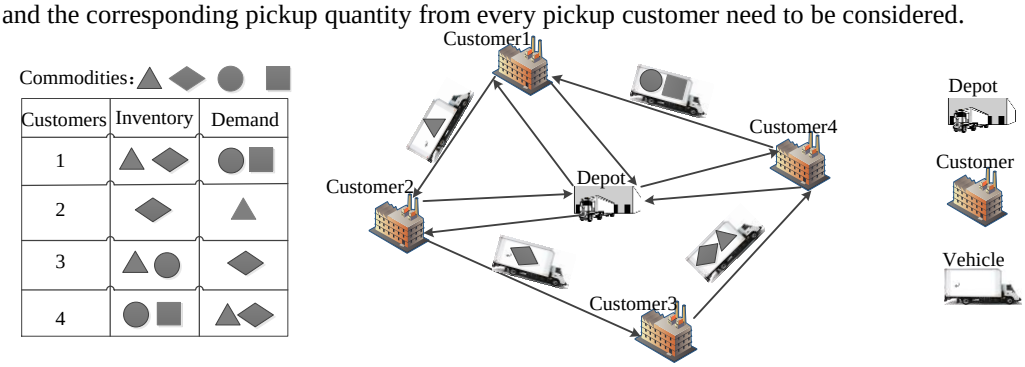


Fig. 1. The commodities transshipment network

In this paper, UPDVRP is formulated within a transportation network involving multiple customers, multiple commodities and a single depot. The supply and demand information of each commodity in each customer is predetermined. A certain quantity of a commodity is collected from its any supply-customer can be delivered to its any demand-customer (Unpaired). Split pickup and delivery of any commodity at a customer are allowed (Split Loads and Unloads). A fleet of vehicles is assigned to pick up and deliver commodities among customers to satisfy every customer's demand. The objective is to construct the optimal routing solution with the minimal transportation costs, subject to duration constraint and capacity constraint. The routing solution should include the visiting sequence of customers of

each transportation vehicle and the quantity of each commodity carried from one customer to another. To solve the proposed UPDVRP, an improved variable neighborhood search algorithm (IVNS) is presented by combining multi-start initial solution generation technique and modified multi-local search operators.

The contributions of this paper are summarized as follows. More compact model and valid inequalities for multi-commodity UPDVRP with split loads and unloads are proposed, which lays a good theoretical foundation for tackling a serial of related problems both in practical and in the literature. Our improved heuristic algorithm can provide optimal or near optimal solutions for the investigated problem quickly. Also, our proposed algorithm is easy to implement and is superior to the existing literature, thus it is expected to provide more powerful decision-making support and guidance for the actual operation of the enterprises.

The rest of the paper is organized as follows. Relevant works of UPDVRP are reviewed in Section 2. Mathematical model is developed in section 3. Section 4 introduces the valid inequalities based on the model structure. An improved variable neighborhood search algorithm is designed in section 5. Section 6 presents the numerical experiments and in-depth discussions. The paper is concluded in Section 7 and possible spaces for future research are also discussed.

2 Literature ReView

Related work of UPDVRP mainly appears at the area of bike re-balancing in the static bike sharing system of the urban area [6-11] and the commodity rebalancing of the fashion retailing chains [3, 5, 12].

In the context of bike re-balancing, within the static bike sharing system, the work is addressed as Static Bike Rebalancing Problem (SBRP). For SBRP, the inventory and demand of bikes in each station are often known before repositioning takes place and remain unchanged during the repositioning operation. A single/multiple capacitated vehicle(s) is/are employed to transport bikes from excess stations to insufficient stations within the repositioning duration, intending to meet user demand. The optimization objective is set according to the actual operation scenario of the bike-sharing system, such as total travel costs, the total unmet demands, the total penalty costs, etc. Benchimol et al. [13] introduce the single-vehicle case of SBRP, which can be classified into two categories: split loads and unloads or non-split loads and unloads, by considering whether the split pickup and delivery are allowed at a station (i.e., whether it is visited by more than once). For single-vehicle SBRP with non-split loads and unloads, Ho and Szeto [7] propose an iterated local search algorithm. Erdoğan et al. [14] develop a branch-and-cut algorithm. Szeto et al. [15] present an enhanced chemical reaction optimization algorithm. For single-vehicle SBRP with split loads and unloads, Chemla et al. [6] propose a hybrid algorithm which combines branch-and-cut with tabu search; Erdoğan et al. [16] develop a branch-and-cut algorithm, which can solve the instances with up to 60 stations; Pal et al. [17] present a hybrid nested large neighborhood search algorithm and Cruz et al. [18] address an iterated local search algorithm. Li et al. [19] investigate a new single-vehicle SBRP in which multiple types of bikes are considered, a hybrid genetic algorithm is designed. The multi-vehicle case of SBRP is formally introduced by Raviv et al. [20], which can also be classified into two categories: split loads and unloads (SLD) or non-split loads and unloads. For multi-vehicle SBRP with non-split loads and unloads, Raviv et al. [20] propose a two-phase exact algorithm; Dell'Amico et al. [21] develop a branch-and-cut algorithm; Kadri et al. [22] explore a branch-and-bound algorithm; Also, some heuristic algorithms are investigated (a destroy and repair algorithm, Dell'Amico et al. [23]; two-phase heuristic approach,You et al. [24], Schuijbroek et al. [25], You [26], etc). For multi-vehicle SBRP with split loads and unloads, Forma et al. [27] present a 3-step math heuristic algorithm; Rainerharbach et al. [28] investigate and compare several heuristic algorithms (including the variable neighborhood descent, greedy randomized adaptive search procedure and variable neighborhood search); Gaspero et al. [29] develop a large neighborhood search algorithm; Bulhões et al. [30] propose a branch-and-cut algorithm, which can solve the problem with up

to 30 stations. In above, all the studies except for li et al. [19] investigate single type of bikes (single-commodity), and only a few other studies focus on split loads and unloads but with single-commodity. Compared with SBRP, this work considers split loads and unloads, multi-commodity and multi-vehicle simultaneously, thus the complexity of the problem is increased in exponential form.

In the context of commodity rebalancing in the fashion retailing chains, the related work is referred to as Unpaired Pickup and Delivery Vehicle Routing Problem (UPDVRP). The single-vehicle case of the UPDVRP is formally introduced by Hernández-Pérez and Salazar-Gonzalez [31, 32], with the name of One-commodity Pickup and Delivery Traveling Salesman Problem (1-PDTSP). 1-PDTSP considers one commodity and calls for a minimum cost tour, which visits each customer exactly once (non-split loads and unloads) and satisfies every customer's demand. The following lists existing work for 1-PDTSP: Hernández-Pérez and Salazar-Gonzalez [33] propose some new valid inequalities and a branch-and-cut algorithm, which can solve instances with up to 100 customers to optimality; Hernández-Pérez et al. [34] develop a hybrid heuristic algorithm; Zhao et al. [35] address a genetic algorithm; Mladenović et al. [36] present a variable neighborhood search based algorithm; Salazar-Gonzalez and Santos-Hernández [37] extend the 1-PDTSP by allowing split loads and unloads and propose a branch-and-cut algorithm, which can solve the instances with up to 33 customers. PDTSP with multi-commodity (m-PDTSP) is introduced by Hernández-Pérez and Salazar-Gonzalez [12]. The proposed branch-and-cut algorithm can solve the problem with up to 30 customers and 3 commodities with non-split loads and unloads. Further, Hernández-Pérez et al. [38] develop a hybrid heuristic approach and Lu et al. [39] design a population algorithm based on randomized tabu thresholding for m-PDTSP with non-split loads and unloads. The multi-vehicle case of UPDVRP, known as the one-commodity pickup and delivery vehicle routing problem (1-PDVRP), is formally introduced by Shi et al. [40]. Lei and Ouyang [5] present a hybrid modeling framework and a lagrange relaxation based algorithm for 1-PDVRP with non-split loads and unloads. Chen et al. [3] introduce m-PDVRP with split loads and unloads by considering unlimited supply (i.e., the inventory of each commodity at supply customer is infinite). A variable neighborhood search algorithm, which can obtain optimal solution or near optimal solution within 1800s, is proposed. Xu et al. [4] study the m-PDVRP with split loads and unloads and limited supply, the proposed tabu search algorithm can provide high-quality solution within 180s for the investigated problem in a short time. This research extends the work of m-PDVRP with split loads and unloads by exploring the improved mathematical model, valid inequalities and solution algorithm.

3 Mathematical Model

3.1 Problem Definition and Assumptions

We the formulate Multi-Commodity Unpaired Pickup and Delivery Vehicle Routing Problem with Split Loads and Unloads(MC-UPDVRP-SLD), based on the transportation network of actual enterprise. Let $G = (N, A)$ be a complete and directed graph, where $N = \{0, 1, 2,\ldots, n, n+1\}$ is the set of vertices. Vertices 0 and $n+1$ denote the departure and destination depots, respectively. Other vertices correspond to customers. $A=\{(i,j):i, j \in N, i \neq j\}$ is the arc set, each arc (i, j) has a non-negative length. Let $H = \{1, 2,\ldots, M\}$ represent the set of commodity types to be transported among customers. Each customer requires a certain quantity of a few types of commodities and holds a certain inventory of some other commodities. Let D_{im} and I_{im} denote the quantity of commodity m required by customer i and the quantity of commodity supplied by customer i, respectively. Let $V = \{1, 2,\ldots, K\}$ be a set of available vehicles. A series of vehicle routes are constructed to pick up and deliver commodities within the network such that:(1) the load and working time of a vehicle never exceeds its capacity; (2) the requirements of all the customers are satisfied by delivery; (3) the total transportation costs is minimal.

By referring to Chen et al. [3], the following assumptions are provided for the convenience of modeling.

(1) The number of transportation vehicle is enough, and all vehicles have the same speed, capacity and maximum working time.

(2) Each vehicle is only allowed to visit a customer once at most.

(3) Loading/unloading time at each customer is neglected.

(4) Arbitrary split for pickup/delivery on a customer is allowed, which means the total demand of a commodity at a customer can be satisfied by several deliveries of different vehicles.

(5) Each vehicle starts from and returns to the depot with empty.

3.2 Model

A mixed integer linear programming model is developed for the investigated UPDVRP. The main difference between our model and the model of Chen et al. [3] lies in that we develop a more compact and effective way to represent "Service Sequence Constraint" and "Loading and Unloading Constraints". The additional symbols and variables used in the model are defined as follows.

f_{im}^k auxiliary parameter, which equals to 1 if vertex *i* supplies commodity *m*, equals to -1 if vertex *i* has demand for commodity *m*, and 0 otherwise;

c_{ij} travel cost between vertices *i* and *j*, which is equal to the travel distance between these two vertices and follows the triangle inequality principle;

t_{ij} travel time between vertices *i* and *j*, which is equal to the ratio of c_{ij} over vehicle speed;

Q capacity of the vehicle;

T maximum working time of the vehicle;

x_{ij}^k equal to1 if vehicle *k* travels from vertex *i* directly to vertex *j*, and 0 otherwise;

q_{im}^k pickup/delivery quantity of commodity *m* at vertex *i* when it is visited by vehicle *k*;

l_{ijm}^k load quantity of commodity *m* on vehicle *k* when it travels from vertex *i* directly to vertex *j*;

y_i^k visiting order of vertex *i* in the route of vehicle *k*.

3.2.1 Objective Function

Objective function (1) minimizes the total transportation costs.

$$Min \sum_{i\in N}\sum_{j\in N}\sum_{k\in V} x_{ij}^k c_{ij} \tag{1}$$

3.2.2 Vehicle Initial and Finishing Position Constraints

Constraints (2) and (3) ensure that each vehicle starts from the departure depot and terminates at the destination depot, respectively. Moreover, $x_{0(n+1)}^k = 1$ indicates that vehicle *k* has no transportation task and stops at the depot.

$$\sum_{j\in N} x_{0j}^k = 1, \quad \forall k \in V \tag{2}$$

$$\sum_{j\in N} x_{j(n+1)}^k = 1, \quad \forall k \in V \tag{3}$$

3.2.3 One-Visit Constraints

Constraint (4) denotes that each customer is visited at most once by a vehicle. Constraint (5) is the flow conservation constraint.

$$\sum_{i\in N} x_{ij}^k \le 1, \quad \forall j \in N \setminus \{0, n+1\}, k \in V \tag{4}$$

$$\sum_{i\in N} x_{ij}^k - \sum_{i\in N} x_{ji}^k = 0, \quad \forall j \in N \setminus \{0, n+1\}, k \in V \tag{5}$$

3.2.4 Service Sequence Constraint

Constraint (6) intends to eliminate sub-tours for each route.

$$y_j^k \ge y_i^k + 1 - (n+1)*(1 - x_{ij}^k), \quad \forall i \in N, j \in N, k \in V \tag{6}$$

3.2.5 Loading and Unloading Constraints

Constraints (7) and (8) denote that if vehicle *k* visits customer *j*, a quantity of commodity *m* onboard the vehicle when it arrives at customer *j*, plus the pickup quantity or minus the delivery quantity of commodity *m* at customer *j*, equals to the quantity of commodity *m* on the vehicle when it departs from customer *j*. Constraint (9) indicates that the total pickup quantity of commodity *m* at customer *j* cannot exceed its inventory. Constraint (10) denotes that the demand of each customer is fully satisfied. Constraint (11) denotes that if vehicle *k* visits customer *j*, the total pickup and delivery quantity of each commodity at each customer cannot exceed its load capacity. In addition, constraint (11) indicates that q_{jm}^k should be zero, if customer *j* is not visited by vehicle *k*. Constraint (12) specifies the capacity of each vehicle. Moreover, constraint (12) indicates that the quantity of each commodity onboard vehicle *k* when it travels from customer *i* to customer *j* is zero, if arc (i, j) is not visited by vehicle *k*. Constraints (13) and (14) specify that each vehicle should be empty when it starts from departure depot and ends at destination depot, respectively.

$$\sum_{i\in N\setminus\{n+1\}} l_{ijm}^k + q_{jm}^k f_{jm} - \sum_{i\in N\setminus\{0\}} l_{jim}^k - Q\sum_{i\in N} x_{ij}^k \ge -Q, \quad \forall j \in N \setminus \{0\}, m \in H, k \in V \tag{7}$$

$$\sum_{i\in N\setminus\{n+1\}} l_{ijm}^k + q_{jm}^k f_{jm} - \sum_{i\in N\setminus\{0\}} l_{jim}^k + Q\sum_{i\in N} x_{ij}^k \le Q, \quad \forall j \in N \setminus \{0\}, m \in H, k \in V \tag{8}$$

$$\sum_{k\in V} q_{jm}^k \le I_{jm}, \quad \forall j \in N \setminus \{0, n+1\}, m \in \{H \mid I_{jm} > 0\} \tag{9}$$

$$\sum_{k\in V} q_{jm}^k = D_{jm}, \quad \forall j \in N \setminus \{0, n+1\}, m \in \{H \mid D_{jm} > 0\} \tag{10}$$

$$q_{jm}^k \le Q\sum_{i\in N} x_{ij}^k, \quad \forall j \in N \setminus \{0, n+1\}, m \in H, k \in V \tag{11}$$

$$\sum_{m\in H} l_{ijm}^k \le Q x_{ij}^k, \quad \forall i \in N, j \in N, k \in V \tag{12}$$

$$l_{0jm}^k = 0, \quad \forall j \in N, m \in H, k \in V \tag{13}$$

$$l_{j(n+1)m}^k = 0, \quad \forall j \in N, m \in H, k \in V \tag{14}$$

3.2.6 Duration Constraint

Constraint (15) denotes that the travel duration of each vehicle cannot exceed its maximum allowing working time.

$$\sum_{i\in N}\sum_{j\in N} x_{ij}^k t_{ij} \le T, \quad \forall k \in V \tag{15}$$

3.2.7 Variable Relationship Constraint

Constraint (16) denotes that if vehicle *k* has no pickup and delivery tasks at customer *j*, then vehicle *k* would not visit customer *j*.

$$\sum_{i \in N} x_{ij}^{k} \le \sum_{m \in H} q_{jm}^{k}, \quad \forall j \in N \setminus \{0, n+1\}, k \in V \tag{16}$$

3.2.8 Variable Type Constraints

$$x_{ij}^{k} \in \{0,1\}, \quad \forall i \in N, j \in N, k \in V \tag{17}$$

$$y_{i}^{k} \in [0, n+1], \quad \forall i \in N, k \in V \tag{18}$$

$$q_{im}^{k} \ge 0, \quad \forall i \in N, m \in H, k \in V \tag{19}$$

$$l_{ijm}^{k} \ge 0, \quad \forall i \in N, \ j \in N, m \in H, k \in V \tag{20}$$

4 Valid Inequalities

Due to the high complexity of the investigated problem, even if the modified model cannot obtain optimal solution for instances with customer number of 7 and commodity number of 4 (see, Table 2 in Section 6.2 of this paper). Therefore, some valid inequalities are proposed to help the model (i.e., by CPLEX) obtain optimal solution or tighter lower bound within prescribed time or before out-of-memory, so as to verify the effectiveness of the heuristic algorithm proposed in Section 5 by comparing its results with the high-quality lower bound derived from CPLEX.

4.1 Least-visiting Times Inequality

By referring to Chen et al. [3], inequality (21) specifies the number of times each customer is visited at least, according to the customer demand and vehicle capacity.

$$\sum_{k \in V} \sum_{i \in N} x_{ij}^{k} \ge \left\lceil \frac{\sum_{m \in H} D_{jm}}{Q} \right\rceil, \quad \forall j \in N \setminus \{0, n+1\} \tag{21}$$

4.2 Vehicle Selecting Inequality

As stated in Section 3.2.2, $x_{0(n+1)}^{k} = 1$ indicates that vehicle k is not used. Thus, inequality (22) ensure that the vehicle with smaller index is dispatched first.

$$x_{0(n+1)}^{k} \le x_{0(n+1)}^{k+1}, \quad \forall k \in V \setminus \{K\} \tag{22}$$

4.3 Arc Eliminating Inequalities

Based on the branch-and-cut feature of CPLEX in solving integer programming (IP) problem[42,43], inequalities (23) ~ (25) are proposed, intending to improve the lower bound of our IP problem by reducing the feasible solution space of the corresponding linear programming (LP) relaxation. For example, inequalities (23) can help CPLEX to filter out some feasible (i.e., for the LP relaxation) fractional solutions that contain $x_{12}^{k} = 0.6$ and $x_{0(n+1)}^{k} = 0.6$.

$$x_{ij}^{k} \le (1 - x_{0(n+1)}^{k}), \quad \forall i, j \in N \setminus \{0, n+1\}, k \in V \tag{23}$$

$$x_{0i}^{k} \le (1 - x_{0(n+1)}^{k}), \quad \forall i \in N \setminus \{0, n+1\}, k \in V \tag{24}$$

$$x_{i(n+1)}^{k} \le (1 - x_{0(n+1)}^{k}), \quad \forall i \in N \setminus \{0, n+1\}, k \in V \tag{25}$$

4.4 Minimum Vehicle Number Required Inequality

Same as other vehicle routing problem, the number of vehicles has a great influence on the solving procedure of this problem, thus we explore the minimum number of vehicles required to exactly solve the problem. Not hard to find, the *Model** with the minimum number of the vehicles required as the objective function can be constructed, by replacing objective function (1) with $Min \sum_{k \in V} (1 - x_{0(n+1)}^{k})$ in the *Model* subject to constraints (2)~(20). Obviously, *Model** has the same solution space as that for *Model* (constraints are identical). Although *Model** is very hard to be solved exactly (it is a NP-hard problem), the lower bound of *Model** (i.e., r_{lb}) can be obtained by solving its corresponding linear relaxation programming. Thus, inequality (26) is valid.

$$\sum_{k \in V} (1 - x_{0(n+1)}^{k}) \ge \lceil r_{lb} \rceil \tag{26}$$

4.5 Solution Refining

During the problem solving procedure, different pairing decisions may results in different transportation scheme but with the identical transportation costs, thus there may be multiple solutions with the same objective value. Fig.2 and Fig.3 present two different transportation schemes for illustrating this phenomenon. The demand of customer *h* is supplied by customer *j* in transportation scheme 1, while it is supplied by customer *p* in transportation scheme 2. The two different supply-demand pairs for demand customer *h* result in different transportation schemes, however with the identical transportation costs as with the same travel routes.

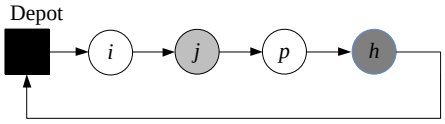


Fig. 2. Transportation Scheme 1

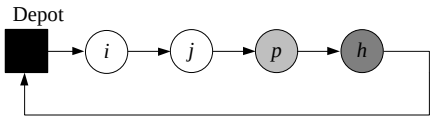


Fig. 3. Transportation Scheme 2

Considering oil consumption in the actual transportation, the smaller value of "load × distance" indicates the better of the transportation scheme. Thus, we introduce $c_{ij} l_{ijm}^{k}$ into the objective function, aiming to refine the solutions with the same objective values. The objective function (1) is accordingly updated to

$$Min \quad \sum_{i \in N} \sum_{j \in N} \sum_{k \in V} c_{ij} x_{ij}^{k} + \psi \sum_{i \in N} \sum_{j \in N} \sum_{k \in V} \sum_{m \in H} c_{ij} l_{ijm}^{k} \tag{27}$$

In the experiments, the weighting parameter ψ is set to $\frac{1}{100 * K * Q \max\limits_{0 \le i, j \le n} \{c_{ij}\}}$, intending to assure that (27) not only can determine a better solution with a lower transportation costs (just like objective function (1)), but also can refine solutions with identical transportation costs by selecting the one with the smallest value of "load × distance" (i.e., it can confirm further a better solution from solutions with identical transportation costs). Preliminary tests show that objective function (27) relatives to (1) can significantly accelerate the solving process. The reason is that the objective function (27) introduces "load × distance" to discard some dominant solutions, which can be regarded as valid inequality to shrink the feasible search space.

5 Improved Variable Neighborhood Search（IVNS）

The variable neighborhood search algorithm (VNS) is a meta-heuristic algorithm, which is proposed by Mladenović and Hansen [41]. It has been successfully applied in solving large scale PDVRP. Chen et al. [3] has proposed a variable neighborhood search algorithm for this investigated problem. Based on Chen et al. [3], this paper design a more efficient VNS by using multi-start initial solution, more efficient neighborhood structure, and more flexible search process.

5.1 Initial Solution Construction

The research objective of this paper is to find the solution (i.e., transportation scheme) with the shortest transportation distance to meet all customer demands, so the total transported quantity of commodities is known in advance, it is equal to the sum of all customer demands. Given a solution, suppose the ratio of the total transportation quantity over the travel distance is denoted as transportation efficiency δ. Intuitively, the lager value of δ, the better of the corresponding solution. Thus, Chen et al. [3] propose a heuristic transportation efficiency-based algorithm (TEBA) to obtain initial solution.

As it is simple and efficient, this paper also applies TEBA to generate initial feasible solution. The key idea of this algorithm is: Firstly, select two customer i and j to construct a starting route: $0-i-j-(n+1)$, and determine the maximum possible pairing quantity between customers i and j from the first commodity subject to the capacity and duration constraints. Then, select a customer and insert into the corresponding position of the current route with the largest value of δ in succession, until a new insertion cannot be found and a feasible route is generated. Lastly, the above process is repeated, until all customers have been inserted and all its demands are satisfied. The specific process of TEBA can refer to Chen et al. [3].

In addition, considering that TEBA may result in different initial feasible solution when choosing different customers i and j to generate starting routes, this paper differs from Chen et al. [3], running TEBA $n*(n-1)$ times by taking different starting route: $0-i-j-(n+1)$ to generate a series of initial solutions. Then, sorting them in the non-decreasing order based on the objective values to obtain an initial feasible solution sequence.

5.2 Neighborhood Solution Generation

The solution consists of supply-demand pairing and routing decisions, these two decisions interact with each other. If the supply-demand pairing decision is changed, the corresponding optimal routing decision is altered, vice versa. Thus, the neighborhood feasible solution can be generated by cross-disturbance to these two decisions. The main processes are given as follows: Firstly, some supply-demand pairs are removed from the current solution. Then, a local search is used to re-insert unsatisfied customers into new positions of the current solution to meet their demands by re-building supply-demand pairs.

5.2.1 Neighborhood Shaking

Based on the integration and improvement of the six neighborhoods proposed by Chen et al. [3], the following four neighborhoods are explored in this paper. Comparing to Chen et al. [3], although the number of neighborhoods is decreased, but the search efficiency is improved by enhancing the intensification and diversification of changing the supply-demand pairs and customer visiting sequence.

(1) Delete some supply-demand pairs.

Randomly delete η supply-demand pairs from the current solution. η is generated by $\mathrm{U}[1, \min(3, \lambda)]$, where λ denotes the number of supply-demand pairs in the current solution.

(2) Delete all the supply-demand pairs of some commodities in one route.

Firstly, a route *r* from the current solution is selected. Then, randomly select m_r types of commodities that route *r* delivered, m_r is generated by $\mathrm{U}\left[1,\ \min\left(3, M_r\right)\right]$, where M_r denotes the number of commodity types that route *r* delivered. Lastly, all the supply-demand pairs related to these m_r types of commodities in route *r* are deleted.

(3) Delete all the supply-demand pairs of one commodity in some routes.

Firstly, randomly select *R* routes from the current solution, *R* is generated by $\mathrm{U}\left[1, \min\left(3, R_c\right)\right]$, where R_c denotes the number of routes in current solution. Then, randomly select commodity *m* related to these selected routes. Lastly, delete all the supply-demand pairs relating to commodity *m* in these *R* routes.

(4) Delete all the supply-demand pairs of one customer.

Firstly, randomly select one customer *i* from the current solution. Then, delete all the supply-demand pairs related to customer *i*. This neighborhood integrates three neighborhoods related to customer in Chen et al. [3], aiming to enable the customer's demand to be satisfied as centrally as possible, so as to improve the current solution by reducing the visiting time of each customer.

5.2.2 Local Search

After removing some supply-demand pairs by executing neighborhood shaking, Chen et al. [3] propose local search procedure to re-insert unsatisfied customers into the current solution, intending to attain locally optimal feasible solution. The main procedure of the local search is as follows: For one unsatisfied customer, firstly search each possible insertion position of every route in the current solution, and record its corresponding increased costs and maximum allowed pairing quantity with the front customers. (see Fig.4 in Chen et al. [3]). Then, a branch-and-bound tree is applied to find the optimal combination of feasible insertion positions to re-insert the customer and meet its demand. (see Fig.5 in Chen et al. [3]). Based on Chen et al. [3], this paper revises the search procedure for the feasible insertion position. The previous and modified search procedures are presented in the following Fig.4 and Fig.5, respectively. The shadow node represents the corresponding customer that can provide the commodities for demand customer *j*. "SC" denotes the spare capacity when the vehicle travels the arc. In a word, when exploring new insertion position of one route to meet the demand of a customer by re-building supply-demand pairs with front other customers, relative to Chen et al. [3], this paper not only considers the customers (i.e., customers *g* and *i* in Fig.5) have been in the route, bust also considers introducing new customer (i.e., customer *d* in Fig.5) into the route. Thus, the better feasible neighborhood solution is expected to be obtained by expanding the search space of feasible insertion positions.

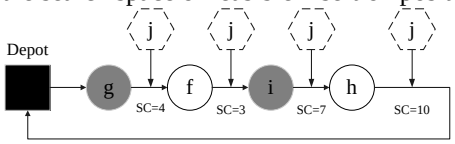


Fig. 4. Route 1 without j

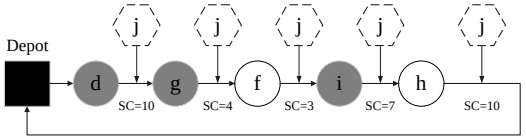


Fig. 5. Route 1 without j

5.3 Algorithm Procedure

Generally speaking, a single initial feasible solution is easy to get the search stuck in local optimal. Whilst an introduced neighborhood operator in a fixed order is easy to cause the

search mechanism inflexible. Moreover, a large number of iterations for each instance would cause useless waste for small instances which optimal solution or near optimal solution can be found quickly. Thus, it is necessary to develop high efficiency algorithm for the proposed UPDVRP. By referring to the work of Chen et al. [3], this paper presented a modified procedure as follows:

1) If all neighborhood operators have performed three times already but the current best solution keeps unchanged, a neighborhood operator is randomly selected to continue.

2) If the current best solution has not been updated after 100 consecutive iterations, a new feasible solution from the initial feasible solution sequence is selected to replace the current solution and restart the search.

3) If the total iteration number reaches up to 1000 times or the initial solution is updated 6 times, the algorithm is stopped.

The specific process of improved variable neighborhood search algorithm is shown in Fig.6.

```
Set F_C the current solution, F_B the best solution, neighborhood number β_max =4;
Set maximum iterate number α_max =1000, maximum number of initial solutions used γ_max =6;
Select the first solution F from initial solution sequence;
Initialization: F_C = F, F_B =F, α=0, γ=0, β=1, θ=0;
While (α < 1000 and γ < γ_max) do
    //Neighborhood Shaking
    Generate F_C' based on F_C by executing the neighborhood N_β(F_C);
    //Local Search
    Local search is applied to F_C' to obtain locally optimal feasible solution F_C'';
    If (F_C'' is better than F_C) then
        Update F_C:=F_C'', β:=1;//restart with the first neighborhood
    Else
        Update β: = β % β_max +1; //execute the next neighborhood
    If (F_C'' is better than F_B ) then
        Update F_B:=F_C'', θ:=0, β:=1;//restart with the first neighborhood
    Else
        Update θ:=θ+1;
        If (θ % (3β_max ))=0) then //F_B is not updated executing all the neighborhoods 3 times
            Update β: = rand(1, β_max );
        If (θ=100) then
            Select a new solution form initial solution sequence as F_C (diversification);
            Update γ: = γ +1, θ:=0;
    α=α+1;
Return the best solution F_B;
```

Fig. 6. The procedure of improved variable neighborhood search

6 Numerical Experiments

In this section, firstly, the model proposed by Chen et al. [3] and the improved model developed in this paper are compared. Then, the effectiveness of the valid inequalities for

strengthening our model is explored. Finally, the performance of the improved variable neighborhood search algorithm (IVNS) designed in this paper is tested.

The experimental environments are given as: (1) all the tests are executed on a computer with Intel dual core 2.8 GHz CPU and 2 GB RAM, and using CPLEX 12.2 as an ILP solver, in order to keep consistent with Chen et al. [3] for fair comparison. (2) The heuristic algorithm is implemented by C++ coding on Visual Studio 2008 platform. For convenience, the model proposed by Chen et al. [3] is regarded as original model and denoted as Model_O, the improved model developed in this paper is denoted as Model_I, the improved model with the valid inequalities is denoted as Model_IV.

6.1 Experimental Instances

18 group instances with different sizes ($n = 5,6,7,8,9,10$; $M = 2,4,6$) in Chen et al. [3] are shared for experiments. Based on which the improved model, valid inequalities and variable neighborhood search algorithm are tested for the proposed UPDVRP. Each group instances is consisted of 5 randomly generated instances with the same size (i.e., the same number of customers and commodity types). Parameters for generating these instances are shown in Table 1.

Table 1 Parameters for generating test data

Parameter	Description of Parameter	Distribution/value
n	The number of customers	5, 6 ,7, 8, 9, 10
M	The number of commodity types	2, 4, 6
T	Vehicle duration	10
Q	Vehicle capacity	10
speed	Vehicle speed	60
(a_i,b_i)	Location of vertex i	$a_i, b_i \sim \text{U}[0, T \times speed / sqt(8)]$
c_{ij}	Travel cost between vertices i and j	$\sqrt{(a_i - a_j)^2 + (b_i - b_j)^2}$
t_{ij}	Travel time between vertices i and j	$c_{ij} / speed$
I_{im}	The inventory of commodity m at customer i	0 with probability 40% or 100000
D_{im}	The demand of commodity m at customer i	if I_{im} = 0, D_{im} = 0 with probability 10%, U[0, 10] with probability 70%, U[0, 20] with probability 20%; if $I_{im} \neq 0$, D_{im} = 0.

6.2 The effectiveness of Model_I

Model_I is used to test the first instance in 18 groups of instances by CPLEX, and the test results are compared with those of Model_O in Chen et al. [3]. The maximum allowed computational time of each instance is 48h, which is in accordance with the settings in Chen et al. [3]. Corresponding test results are reported in Table 2. In Table 2, the first column denotes the instance name, which is made up of customer number and commodity type number. Columns 2~7 represent the results of Model_O and Columns 8~13 denotes the results of Model_I. "LB" and "UB" are the lower and upper bounds of the instance, respectively. "Time" refers to the time spent on solving the instance via CPLEX. "Gap" denotes the percent difference between LB and UB, which is computed as $\text{Gap} = 100 \times (\text{UB} - \text{LB}) / \text{LB}$. "Var" and "Cons" denote the number of decision variables and constraints involved in the model, respectively. "$\Delta 1$" denotes the percent difference between lower bounds obtained by Model_O and Model_I, "$\Delta 2$" denotes the percent difference between decision variable number by Model_O and Model_I, and "$\Delta 3$" denotes the percent difference between constraint number by Model_O and Model_I.

Table 2 The Comparison of $Model_O$ 和 $Model_I$

Instances	$Mode1_O$						$Model_I$						Gap(%)		
	LB	UB	Time(s)	Gap	Vars	Cons	LB	UB	Time(s)	Gap	Vars	Cons	Δ1	Δ2	Δ3
5n2M_1	**391.46**	**391.46**	1.93	0	889	2683	**391.46**	**391.46**	2.95	0	473	1072	0	-46.79	-60.04
6n2M_1	**957.66**	**957.66**	104.98	0	2689	8621	**957.66**	**957.66**	20.35	0	1265	2808	0	-52.96	-67.43
7n2M_1	**977.80**	**977.80**	1877.45	0	7809	26118	**977.80**	**977.80**	447.69	0	3265	7148	0	-58.19	-72.63
8n2M_1	**817.20**	**817.20**	1256.43	0	5473	18878	**817.20**	**817.20**	229.57	0	2048	4464	0	-62.58	-76.35
9n2M_1	**921.19**	**921.19**	24518.28	0	13951	49237	**921.19**	**921.19**	8592.77	0	4711	10161	0	-66.23	-79.36
10n2M_1	757.85	829.47	11480.23	9.45	18481	66383	**829.47**	**829.47**	2865.35	0	5671	12160	9.45	-69.31	-81.68
5n4M_1	**940.54**	**940.54**	579.86	0	993	3321	**940.54**	**940.54**	24.64	0	721	1764	0	-27.39	-46.88
6n4M_1	**1055.69**	**1055.69**	4174.60	0	1465	5166	**1055.69**	**1055.69**	10.29	0	977	2336	0	-33.31	-54.78
7n4M_1	844.87	1574.95	3124.12	86.41	5221	19072	1160.98	1481.13	4035.85	27.58	3181	7386	37.42	-39.07	-61.27
8n4M_1	1076.62	2115.84	2731.50	96.53	5055	18934	1622.67	1828.34	7357.81	12.67	2815	6462	50.72	-44.31	-65.87
9n4M_1	869.17	1484.41	4845.33	70.78	5833	22240	**1161.22**	**1161.22**	2428.42	0	2977	6762	33.6	-48.96	-69.6
10n4M_1	581.11	1371.72	3229.69	136.05	10225	39477	802.44	1207.63	2824.02	50.49	4801	10768	38.09	-53.05	-72.72
5n6M_1	1746.87	1888.21	11378.76	8.09	5207	18745	**1792.03**	**1792.03**	263.50	0	4599	11441	2.59	-11.68	-38.97
6n6M_1	1607.49	2097.38	26819.56	30.48	14257	54050	1776.65	1979.76	4198.39	11.43	11881	28728	10.52	-16.67	-46.85
7n6M_1	1424.77	1737.88	15915.46	21.98	16681	65414	1299.75	1631.09	2594.42	25.49	12961	30714	8.77	-22.3	-53.05
8n6M_1	1383.11	1682.29	18342.62	21.63	21281	85334	1051.18	1578.32	5211.12	50.15	15345	35768	-24	-27.89	-58.08
9n6M_1	1495.85	2794.12	36248.29	86.79	5071	20653	1544.66	2563.19	521.95	65.94	4069	9438	3.26	-19.76	-54.3
10n6M_1	1128.19	2017.6	15601.49	78.84	46341	190453	1162.03	1467.33	2492.60	26.27	4111	9440	3	-91.13	-95.04
Avg.	1054.30				10384.5		1125.81				4770		8.66	-54.06	-64.16

The test results in Table 2 show that the complexity of the problem investigated in this paper increases exponentially with the problem size getting larger (i.e., the number of customers and the number of commodity types). With the size of the problem increases, CPLEX is forced to stop before time is up to the specified value for the computer has already run out of memory. Within the 48 hours, $Model_I$ can reach optimality at three more instances comparing to $Model_O$. For the seven instances that can be solved optimally by $Model_O$ and $Model_I$, the average time spent by $Model_O$ and $Model_I$ is 4644.79 seconds and 1322.61 seconds, compared to $Model_O$, $Model_I$ can save computing time by 71.31%. For the eight instances that cannot be solved optimally, $Model_I$ can obtain the tighter lower bound, which is improved by 13.78% on average and by 50.72% on maximum comparing to $Model_O$. Besides, columns "Δ2" and "Δ3" indicates that $Model_I$ can save the number of decision variables and constraints by 54.06% and 64.16% on average respectively, which leads to $Model_I$ more powerful than $Mode_O$. The above results show that $Model_I$ is a more concise model, which not only lay a good foundation for theoretical research on UPDVRPs, but also provide theoretical guidance for practical operation of enterprises.

6.3 The performance of valid inequalities

To verify the performance of valid inequalities, $Model_{IV}$ is tested by the above 18 instances (i.e., in Table 2), and the test results are compared with those of $Model_I$, as shown in Table 3. Column titles "LB", "UB" and "Time" represent the same meaning as those in Table 2. Column "Node" denotes the number of explored nodes in branch-bound tree. Columns "$\Delta 1$", "$\Delta 2$" and "$\Delta 3$" denote percent differences between lower bounds, computation time, explored nodes by $Model_I$ and $Model_{IV}$, respectively. It should point out that $Model_{IV}$ is performed with objective function (27), relative to $Model_I$ with objective function (1), the second term "load $\times$ distance" is added. To keep consistent with $Model_I$, only first term of objective value obtained by $Model_{IV}$ is displayed in the following Tables.

Table 3 The Comparison of $Model_I$ and $Model_{IV}$

Instances	$Model_I$					$Model_{IV}$					Gap(%)		
	LB	UB	Time(s)	Node	Gap	LB	UB	Time(s)	Node	Gap	$\Delta 1$	$\Delta 2$	$\Delta 3$
5n2M_1	**391.46**	**391.46**	2.95	140	0	**391.46**	**391.46**	2.13	0	0	0	-27.8	-100
6n2M_1	**957.66**	**957.66**	20.35	20641	0	**957.66**	**957.66**	2.92	326	0	0	-85.65	-98.42
7n2M_1	**977.80**	**977.80**	447.69	286436	0	**977.80**	**977.80**	13.55	1678	0	0	-96.97	-99.41
8n2M_1	**817.20**	**817.20**	229.57	199828	0	**817.20**	**817.20**	24.16	2095	0	0	-89.48	-98.95
9n2M_1	**921.19**	**921.19**	8592.77	1687283	0	**921.19**	**921.19**	99.40	49107	0	0	-98.84	-97.09
10n2M_1	**829.47**	**829.47**	2865.35	796037	0	**829.47**	**829.47**	23.91	11564	0	0	-99.17	-98.55
5n4M_1	**940.54**	**940.54**	24.64	53528	0	**940.54**	**940.54**	1.49	1696	0	0	-93.95	-96.83
6n4M_1	**1055.69**	**1055.69**	10.29	10308	0	**1055.69**	**1055.69**	13.40	8913	0	0	30.22	-13.53
7n4M_1	1160.98	1481.13	4035.85	642076	27.58	**1481.13**	**1481.13**	162.36	83520	0	27.58	-95.98	-86.99
8n4M_1	1622.67	1828.34	7357.81	1107352	12.67	**1818.53**	**1818.53**	1072.46	400238	0	12.07	-85.42	-63.86
9n4M_1	**1161.22**	**1161.22**	2428.42	248489	0	**1161.22**	**1161.22**	121.83	36760	0	0	-94.98	-85.21
10n4M_1	802.44	1207.63	2824.02	814153	50.49	**1206.49**	**1206.49**	628.25	702619	0	50.35	-77.75	-13.7
5n6M_1	**1792.03**	**1792.03**	263.50	213322	0	**1792.03**	**1792.03**	14.75	28877	0	0	-94.4	-86.46
6n6M_1	1776.65	1979.76	4198.39	481578	11.43	**1979.48**	**1979.48**	236.60	31843	0	11.42	-94.36	-93.39
7n6M_1	1299.75	1631.09	2594.42	368912	25.49	**1599.09**	**1599.09**	169.07	41076	0	23.03	-93.48	-88.87
8n6M_1	1051.18	1578.32	5211.12	317560	50.15	**1520.18**	**1520.18**	139.44	64072	0	44.62	-97.32	-79.82
9n6M_1	1544.66	2563.19	521.95	58175	65.94	2087.01	2261.45	436.75	20536	8.77	35.76	-16.32	-64.7
10n6M_1	1162.03	1467.33	2492.60	399421	26.27	1386.42	1466.91	5324.66	143095	11.01	19.31	113.62	-64.17
Avg.	1125.81					1274.03					13.17	-66.56	-79.44

The results in Table 3 show that, most of instances (16/18) can be solved to optimality by $Model_{IV}$, compared with $Model_I$, $Model_{IV}$ can solve six more instances to optimality. For the ten instances solved optimally by both $Model_I$ and $Model_{IV}$, $Model_{IV}$ saves the computation time and the explored nodes by 97.87% and 95.99%, respectively. Furthermore, $Model_{IV}$ can obtain tighter lower bound, which is improved by 13.17% on average and by 50.35% on the maximum. The results in Table 3 demonstrate that the valid inequalities can help the model obtain optimal solution or tighter lower bound of the proposed UPDVRP.

6.4 The test results of improved variable neighborhood search algorithm

Table 4 Test results for two types of commodities

算例	VNS			IVNS			CPLEX($Model_{IV}$)			Gap(%)			
	VNS_B	VNS_A	T_1(s)	$AVNS_B$	$AVNS_A$	T_2(s)	LB	UB	Time(s)	Δ1	Δ2	Δ3	Δ4
5n2M_1	**391.46**	**391.46**	96.31	**391.46**	**391.46**	14.73	**391.46**	**391.46**	2.13	0	0	-84.71	0
5n2M_2	**965.79**	**965.79**	196.66	**965.79**	**965.79**	21.7	**965.79**	**965.79**	2.50	0	0	-88.97	0
(*)5n2M_3	854.60	854.60	208.72	**819.40**	**819.40**	23.19	**819.40**	**819.40**	3.25	0	-4.12	-88.89	0
5n2M_4	**264.33**	**264.33**	67.19	**264.33**	**264.33**	3.09	**264.33**	**264.33**	2.08	0	0	-95.4	0
5n2M_5	**1217.84**	**1217.84**	216.88	**1217.84**	**1217.84**	19.66	**1217.84**	**1217.84**	4.98	0	0	-90.94	0
Avg.	738.8	738.8	157.15	731.76	731.76	16.47	731.76	731.76	2.99	0	-0.82	-89.78	0
6n2M_1	**957.66**	**957.66**	156.30	**957.66**	**957.66**	25.58	**957.66**	**957.66**	2.92	0	0	-83.63	0
6n2M_2	**1045.22**	**1045.22**	153.94	**1045.22**	**1045.22**	18.88	**1045.22**	**1045.22**	3.15	0	0	-87.74	0
6n2M_3	**443.83**	**443.83**	86.91	**443.83**	**443.83**	15.19	**443.83**	**443.83**	2.83	0	0	-82.52	0
(*)6n2M_4	746.28	746.28	128.94	**743.62**	**743.62**	23.75	**743.62**	**743.62**	3.48	0	-0.36	-81.58	0
6n2M_5	**204.49**	**204.49**	50.77	**204.49**	**204.49**	10.22	**204.49**	**204.49**	1.97	0	0	-79.87	0
Avg.	679.5	679.5	115.37	678.96	678.96	18.72	678.96	678.96	2.87	0	-0.07	-83.07	0
7n2M_1	**977.80**	**977.80**	291.62	**977.80**	**977.80**	33.72	**977.80**	**977.80**	13.85	0	0	-88.44	0
7n2M_2	**701.91**	**701.91**	190.56	**701.91**	**701.91**	28.78	**701.91**	**701.91**	2.14	0	0	-84.9	0
(*)7n2M_3	1202.87	1202.87	416.45	**1139.35**	**1139.35**	35.36	**1139.35**	**1139.35**	4.50	0	-5.28	-91.51	0
(+)7n2M_4	910.87	910.87	281.17	856.28	867.79	26.61	**838.05**	**838.05**	3.05	2.18	-5.99	-90.54	-1.34
7n2M_5	**716.15**	**716.15**	161.08	**716.15**	**716.15**	21.86	**716.15**	**716.15**	3.02	0	0	-86.43	0
Avg.	901.92	901.92	268.18	878.3	880.6	29.27	874.65	874.65	5.31	0.44	-2.25	-88.36	-0.27
8n2M_1	**817.20**	**817.20**	222.91	**817.20**	**817.20**	31.13	**817.20**	**817.20**	24.16	0	0	-86.03	0
(+)8n2M_2	956.11	957.69	259.17	955.07	955.90	36.97	**940.56**	**940.56**	6.05	1.54	-0.11	-85.74	-0.09
8n2M_3	**739.58**	758.11	242.13	**739.58**	741.38	35.31	**739.58**	**739.58**	3.28	0	0	-85.42	-0.24
8n2M_4	**756.79**	**756.79**	181.03	**756.79**	**756.79**	32.39	**756.79**	**756.79**	3.05	0	0	-82.11	0
8n2M_5	**803.36**	**803.36**	149.72	**803.36**	**803.36**	17.7	**803.36**	**803.36**	4.68	0	0	-88.18	0
Avg.	814.61	818.63	210.99	814.4	814.93	30.7	811.5	811.5	8.24	0.31	-0.02	-85.5	-0.07
(*)9n2M_1	970.49	993.348	272.69	**921.19**	938.01	37.11	**921.19**	**921.19**	99.40	0	-5.08	-86.39	-1.83
(*)9n2M_2	897.08	900.152	328.69	**879.37**	891.70	44.53	**879.37**	**879.37**	66.65	0	-1.97	-86.45	-1.4
(*)9n2M_3	808.32	831.79	1707.05	**805.82**.	807.32	41.92	**805.82**	**805.82**	44.37	0	-0.31	-97.54	-0.19
9n2M_4	**716.55**	**716.55**	196.75	**716.55**	**716.55**	17.41	**716.55**	**716.55**	3.61	0	0	-91.15	0
9n2M_5	**425.33**	**425.33**	179.39	**425.33**	**425.33**	29.33	**425.33**	**425.33**	3.40	0	0	-83.65	0
Avg.	763.55	773.43	536.91	749.65	755.78	34.06	749.65	749.65	43.49	0	-1.47	-89.04	-0.68
(*)10n2M_1	849.07	873.55	365.14	**829.47**	**829.47**	42.13	**829.47**	**829.47**	23.91	0	-2.31	-88.46	0
10n2M_2	**426.0**1	429.19	172.53	**426.0**1	**426.0**1	26.36	**426.01**	**426.01**	13.08	0	0	-84.72	0
10n2M_3	**1715.81**	**1715.81**	291.56	**1715.81**	**1715.81**	24.92	**1715.81**	**1715.81**	19027.92	0	0	-91.45	0
(+)10n2M_4	747.45	747.45	225.63	743.06	743.06	28.94	**742.24**	**742.24**	104.29	0.11	-0.59	-87.17	0
10n2M_5	**1774.84**	**1774.84**	965.92	**1774.84**	**1774.84**	34.52	**1774.84**	**1774.84**	2481.39	0	0	-96.43	0
Avg.	1102.64	1108.17	404.16	1097.84	1097.84	31.37	1097.67	1097.67	4330.12	0.02	-0.58	-89.65	0

To assess the performance of IVNS, all the instances shown in Section 6.1 are tested by the heuristic algorithm (executed five times for each instance) and CPLEX ($Model_{IV}$), and the results are compared with those in Chen et al. [3]. The detailed results are shown in Tables 4~6 with 2, 4 and 6 types of commodities respectively. "VNS" represents the test results of variable neighborhood search algorithm in Chen et al. [3], "IVNS" represents the test results of improved variable neighborhood search algorithm in this paper, "CPLEX" denotes the results of $Model_{IV}$ in this paper. Column titles "VNS_B", "VNS_A" and "T_1" represent the best solution, average solution and the time consumption of obtaining the best solution with the VNS algorithm executed five times. Column titles "$IVNS_B$", "$IVNS_A$" and "T_2" represent the best solution, average solution and the time consumption of obtaining the best solution with the IVNS algorithm executed five times. Column titles "LB", "UB" and "Time" represent the

lower bound, upper bound and computational time with CPLEX. Column titles "$\Delta 1$", "$\Delta 2$", "$\Delta 3$" and "$\Delta 4$" display respectively the percent differences between $IVNS_B$ and LB, $IVNS_B$ and VNS_B, T_2 and T_1, $IVNS_B$ and $IVNS_A$. In particular, instances marked with "*" represent the instances whose optimal solutions can be obtained by our IVNS algorithm but cannot be obtained by the VNS algorithm in Chen et al. [3]; instances marked with "+" represent the instances whose solutions obtained by our IVNS, although not optimal, are better (with smaller objective value) than those obtained by VNS algorithm in Chen et al. [3]; instances marked with "-" represent the instances whose solutions obtained by our IVNS algorithm are worse than those obtained by VNS algorithm in Chen et al. [3].

Table 5 Test results for four types of commodities

算例	VNS			IVNS			CPLEX($Model_{IV}$)			Gap(%)			
	VNS_B	VNS_A	T_1(s)	$AVNS_B$	$AVNS_A$	T_2(s)	LB	UB	Time(s)	$\Delta 1$	$\Delta 2$	$\Delta 3$	$\Delta 4$
5n4M_1	**940.54**	947.79	917.19	**940.54**	**940.54**	50.19	**940.54**	**940.54**	4.49	0	0	-94.53	0
5n4M_2	**721.21**	**721.21**	176.73	**721.21**	**721.21**	31.22	**721.21**	**721.21**	2.98	0	0	-82.33	0
(*)5n4M_3	879.91	882.55	458.61	**870.56**	**870.56**	37.49	**870.56**	**870.56**	4.19	0	-1.06	-91.83	0
5n4M_4	**1192.61**	**1192.61**	198.72	**1192.61**	**1192.61**	18.2	**1192.61**	**1192.61**	3.57	0	0	-90.84	0
(*)5n4M_5	1694.97	1697.52	468.95	**1681.73**	1688.39	34.34	**1681.73**	**1681.73**	7.47	0	-0.78	-92.68	-0.4
Avg.	1085.85	1088.34	444.04	1081.33	1082.66	34.29	1081.33	1081.33	4.54	0	-0.37	-90.44	-0.08
6n4M_1	**1055.69**	**1055.69**	318.26	**1055.69**	**1055.69**	43.94	**1055.69**	**1055.69**	13.40	0	0	-86.19	0
(*)6n4M_2	1202.44	1278.73	1631.59	**1147.30**	1150.07	46.81	**1147.30**	**1147.30**	5.44	0	-4.59	-97.13	-0.24
6n4M_3	**717.97**	**717.97**	231.78	**717.97**	**717.97**	22.25	**717.97**	**717.97**	3.90	0	0	-90.4	0
6n4M_4	**2117.46**	2119.76	668.19	**2117.46**	**2117.46**	39.5	**2117.46**	**2117.46**	73.21	0	0	-94.09	0
6n4M_5	**906.73**	**906.73**	247.92	**906.73**	**906.73**	22.61	**906.73**	**906.73**	4.67	0	0	-90.88	0
Avg.	1200.06	1215.78	619.55	1189.03	1189.58	35.02	1189.03	1189.03	20.12	0	-0.92	-91.74	-0.05
7n4M_1	**1481.13**	1490.29	507.09	**1481.13**	**1481.13**	42.49	**1481.13**	**1481.13**	162.38	0	0	-91.62	0
(*)7n4M_2	1598.66	1598.66	842.45	**1583.67**	1586.75	54.03	**1583.67**	**1583.67**	8.03	0	-0.94	-93.59	-0.19
(*)7n4M_3	1261.30	1279.51	498.70	**1257.68**	**1257.68**	34.27	**1257.68**	**1257.68**	9.88	0	-0.29	-93.13	0
(*)7n4M_4	1010.84	1024.93	689.65	**999.28**	1004.96	41.00	**999.28**	**999.28**	14.57	0	-1.14	-94.05	-0.57
(+)7n4M_5	1356.00	1446.24	1254.27	1319.11	1344.29	61.77	**1303.91**	**1303.91**	195.47	1.17	-2.72	-95.08	-1.91
Avg.	1341.59	1367.93	758.43	1328.17	1334.96	46.71	1325.13	1325.13	78.07	0.23	-1.02	-93.49	-0.53
(+)8n4M_1	1990.14	1990.14	629.41	1829.02	1834.56	46.89	**1818.53**	**1818.53**	1072.46	0.58	-8.1	-92.55	-0.3
(-)8n4M_2	1487.84	1492.72	886.50	1493.44	1493.66	52.23	**1449.90**	**1449.90**	924.83	3	0.38	-94.11	-0.01
8n4M_3	**866.00**	**866.00**	295.70	**866.00**	**866.00**	24.14	**866.00**	**866.00**	3.96	0	0	-91.84	0
(+)8n4M_4	1540.76	1615.56	454.13	1501.36	1501.86	62.7	**1417.18**	**1417.18**	168.52	5.94	-2.56	-86.19	-0.03
8n4M_5	1482.52	1510.44	1800.00	1482.52	1487.80	48.27	**1474.48**	**1474.48**	8.84	0.55	0	-97.32	-0.36
Avg.	1473.45	1494.97	813.15	1434.47	1436.78	46.85	1405.22	1405.22	435.72	2.01	-2.06	-92.4	-0.14
(*)9n4M_1	1174.41	1232.57	995.83	**1161.22**	1201.91	54.27	**1161.22**	**1161.22**	121.83	0	-1.12	-94.55	-3.5
9n4M_2	**876.26**	**876.26**	261.39	**876.26**	**876.26**	44.39	**876.26**	**876.26**	17.61	0	0	-83.02	0
9n4M_3	1466.36	1466.36	332.48	1466.36	1466.36	40.00	**1466.10**	**1466.10**	28.79	0.02	0	-87.97	0
9n4M_4	**876.23**	876.42	443.66	**876.23**	**876.23**	43.14	**876.23**	**876.23**	44.23	0	0	-90.28	0
9n4M_5	**942.56**	**942.56**	300.89	**942.56**	**942.56**	34.05	**942.56**	**942.56**	25.76	0	0	-88.68	0
Avg.	1067.16	1078.83	466.85	1064.53	1072.66	43.17	1064.47	1064.47	47.64	0	-0.22	-88.9	-0.7
10n4M_1	1218.42	1227.53	1095.94	1218.42	1220.42	66.28	**1206.49**	**1206.49**	628.25	0.99	0	-93.95	-0.16
(-)10n4M_2	1098.39	1119.78	505.90	1110.79	1118.04	54.89	**1094.28**	**1094.28**	9.38	1.51	1.13	-89.15	-0.65
(+)10n4M_3	1534.85	1561.61	1472.17	1488.95	1488.95	49.7	**1450.38**	**1450.38**	30.15	2.66	-2.99	-96.62	0
(-)10n4M_4	1291.41	1326.59	1123.61	1338.66	1348.68	70.23	**1256.17**	**1256.17**	1326.54	6.57	3.66	-93.75	-0.75
(+)10n4M_5	1183.94	1243.35	916.27	1173.4	1173.4	77.00	**1126.16**	**1126.16**	768.13	4.19	-0.89	-91.6	0
Avg.	1265.4	1295.77	1022.78	1266.04	1269.9	63.62	1226.7	1226.7	552.49	3.18	0.18	-93.01	-0.31

Table 6 Test results for six types of commodities

算例	VNS			IVNS			CPLEX($Model_{IV}$)			Gap(%)			
	VNS_B	VNS_A	T_1(s)	$AVNS_B$	$AVNS_A$	T_2(s)	LB	UB	Time(s)	Δ1	Δ2	Δ3	Δ4
(+)5n6M_1	1863.63	1863.63	258.75	1806.59	1825.86	23.31	**1792.03**	**1792.03**	14.75	0	-3.06	-90.99	-1.07
5n6M_2	1833.76	1842.63	1800.00	1833.76	1833.76	46.19	**1820.74**	**1820.74**	92.80	0.72	0	-97.43	0
5n6M_3	**854.98**	**854.98**	179.88	**854.98**	**854.98**	15.70	**854.98**	**854.98**	2.82	0	0	-91.27	0
(+)5n6M_4	2515.81	2572.87	1168.95	2488.2	2516.40	26.87	**2478.48**	**2478.48**	86.67	0.39	-1.1	-97.7	-1.13
(*)5n6M_5	696.94	696.94	147.50	**673.93**	**673.93**	15.84	**673.93**	**673.93**	2.98	0	-3.3	-89.26	0
Avg.	1553.02	1566.21	711.02	1531.49	1540.99	25.58	1524.03	1524.033	40	0.22	-1.49	-93.33	-0.44
(-)6n6M_1	2050.72	2154.84	690.70	2058.42	2074.83	57.60	**1979.48**	**1979.48**	236.60	3.99	0.38	-91.66	-0.8
(+)6n6M_2	1594.37	1594.37	331.11	1589.72	1589.72	49.08	**1529.90**	**1529.90**	6.00	3.91	-0.29	-85.18	0
(+)6n6M_3	1609.80	1616.55	837.59	1591.12	1593.97	56.39	**1588.52**	**1588.52**	43.21	0.16	-1.16	-93.27	-0.18
6n6M_4	1929.87	2019.70	595.39	1929.87	1935.36	62.10	**1894.48**	**1894.48**	7.36	1.87	0	-89.57	-0.28
6n6M_5	**1624.59**	1691.72	489.61	**1624.59**	1662.96	50.44	**1624.59**	**1624.59**	4.89	0	0	-89.7	-2.36
Avg.	1761.87	1815.44	588.88	1758.74	1771.37	55.12	1723.39	1723.39	59.61	1.99	-0.21	-89.88	-0.72
(-)7n6M_1	1646.10	1746.59	690.56	1665.63	1698.56	48.88	**1599.09**	**1599.09**	169.07	4.16	1.19	-92.92	-1.98
(+)7n6M_2	1338.40	1372.48	698.94	1323.16	1331.76	60.73	**1301.67**	**1301.67**	8.56	1.65	-1.14	-91.31	-0.65
7n6M_3	**932.02**	937.43	393.19	**932.02**	**932.02**	45.53	**932.02**	**932.02**	6.06	0	0	-88.42	0
7n6M_4	2029.51	2079.58	440.28	2029.51	2029.51	43.86	**2014.07**	**2014.07**	4.35	0.77	0	-90.04	0
(+)7n6M_5	2362.40	2492.81	1800.00	2297.64	2306.00	73.02	**2271.88**	**2271.88**	199.48	1.13	-2.74	-95.94	-0.36
Avg.	1661.69	1725.78	804.59	1649.59	1659.57	54.4	1623.75	1623.75	77.5	1.54	-.054	-91.73	-0.6
(+)8n6M_1	1611.23	1611.23	407.92	1562.74	1571.33	67.03	**1520.18**	**1520.18**	139.44	2.8	-3.01	-83.57	-0.55
(+)8n6M_2	1567.85	1727.15	879.34	1545.96	1590.64	80.31	**1543.03**	**1543.03**	416.47	0.19	-1.4	-90.87	-2.89
(+)8n6M_3	1698.45	1702.49	1342.33	1658.74	1678.59	66.66	**1623.46**	**1623.46**	396.64	2.17	-2.34	-95.03	-1.2
(+)8n6M_4	1715.87	1728.14	518.77	1691.47	1705.45	67.16	**1656.40**	**1656.40**	89.36	2.12	-1.42	-87.05	-0.83
(+)8n6M_5	2229.51	2229.51	656.28	2228.22	2228.74	66.83	**2143.94**	**2143.94**	226.74	3.93	-0.06	-89.82	-0.02
Avg.	1764.58	1799.7	760.93	1737.43	1754.95	69.6	1697.4	1697.4	253.73	2.24	-1.65	-89.27	-1.1
(*)9n6M_1	2092.35	2209.89	1451.13	**2087.01**	2103.24	73.97	2087.01	2261.45	436.75	0	-0.26	-94.9	-0.78
9n6M_2	1829.52	1893.52	702.86	1829.52	1883.52	65.94	**1825.72**	**1825.72**	536.73	0.21	0	-90.62	-2.95
9n6M_3	**841.62**	**841.62**	246.03	**841.62**	**841.62**	34.94	**841.62**	**841.62**	3.61	0	0	-85.8	0
9n6M_4	**2794.65**	2837.10	687.27	**2794.65**	2813.43	49.65	**2794.65**	**2794.65**	3268.215	0	0	-92.78	-0.67
9n6M_5	**1219.95**	**1219.95**	434.36	**1219.95**	**1219.95**	48.59	**1219.95**	**1219.95**	26.10	0	0	-88.81	0
Avg	1755.62	1800.42	704.33	1754.55	1772.35	54.62	1753.79	1788.68	854.28	0.04	-0.05	-90.58	-0.88
(+)10n6M_1	1514.72	1612.11	1800.00	1495.72	1504.77	97.73	1386.42	1466.91	5324.66	7.88	-1.25	-94.57	-0.61
(+)10n6M_2	2120.34	2333.15	1666.61	2096.14	2260.68	63.28	**2008.59**	**2008.59**	716.29	4.36	-1.14	-96.2	-7.85
10n6M_3	**970.06**	973.61	585.55	**970.06**	971.84	98.45	**970.06**	**970.06**	328.49	0	0	-83.19	-0.18
(*)10n6M_4	2332.01	2332.01	755.53	**2123.40**	2152.83	67.91	**2123.40**	**2123.40**	640.84	0	-8.95	-91.01	-1.39
(+)10n6M_5	2030.91	2114.20	626.72	1997.88	2038.77	59.72	**1915.44**	**1915.44**	1792.54	4.3	-1.63	-90.47	-2.05
Avg.	1793.61	1873.02	1086.88	1736.64	1785.78	77.42	1680.78	1696.88	1760.56	3.31	-2.59	-91.09	-2.42

Based on the results in Tables 4–6, the following findings are presented:

Optimal solutions for most of the instances (82 out of 90) can be obtained by CPLEX($Model_{IV}$) within 48h. However, the optimal solutions of 57 instances (i.e., 27 instances in Table 4, 19 instances in Table 5 and 11 instances in Table 6) can be obtained by IVNS algorithm within just 100s. For the remaining 33 instances, even though the optimal solutions cannot be obtained by IVNS, the percent difference between the obtained solution and the lower bound, as shown in column "$\Delta 1$", is within 8%. Further, CPLEX($Model_{IV}$) fails rapidly to provide the optimal solutions when the instance size is getting larger, while the proposed IVNS can provide high quality solutions for all the instances within 100s. The results clearly show that IVNS can provide optimal or near optimal solutions to the problem presented by Chen et al. [3].

For all the 90 instances, 37 optimal solutions can be obtained by the VNS algorithm in Chen et al. [3] within 1800s. However, our IVNS algorithm not only can obtain the optimal solutions for these 37 instances within just 100 s, but also can obtain optimal solutions for 20 additional instances marked with "*". Further, improved solutions can be provided for 22 instances marked with "+" by IVNS. Only in 5 instances, slightly inferior solutions are obtained, which are marked with "-". In addition, compared with the VNS algorithm in Chen et al. [3], IVNS can improve the solution quality (refer to column "$\Delta 2$") by 1.84% on average and 8.95% to the most; can save computational time (refer to column "$\Delta 3$") by 90.69% on average and 97.32% to the most; can obtain an excellent robustness as the percent difference between INS_B and INS_A is within 4% (refer to column "$\Delta 4$").

From the above, our IVNS algorithm can provide optimal or close to optimal solution for the investigated problem in a short time, and it is superior to the VNS algorithm proposed in Chen et al. [3] in terms of solution quality, computational time and robustness. The main reason why the algorithm has good performance is using the multiple initial feasible solution strategy, more efficient neighborhood operators and more flexible search mechanism. In this paper, the improved algorithm can offer more valuable guideline for the enterprise's actual operation decisions, help enterprises to improve operation efficiency and reduce operating costs, thus promote enterprise's logistics management transition to the refinement and intelligence.

7 Conclusions

Based on practical background of commodity allocation among customers, this paper studies the multi-commodity UPDVRP with split loads and unloads, which is relative scarce in the literature. Firstly, a more concise mixed integer programming model is established by introducing more compact and effective "service sequence constraint" and "loading and unloading constraints". Then, through analyzing the characteristics of the problem, a series of valid inequalities are proposed to strengthen the model. Lastly, by embedding multiple initial feasible solutions, integrating efficient neighborhood exploring operators and considering more flexible search mechanism, an improved variable neighborhood search algorithm is designed to obtain high-quality feasible solutions. The proposed model, valid inequalities and variable neighborhood search algorithm are tested based on the instances in the literature, and the performance results are compared with those of Chen et al. [3].

The experiment results show that: 1) The model proposed in this paper requires less decision variables and constraints, and can optimally solve more instances within prescribed time. 2) The derived valid inequalities can significantly help the model to obtain the optimal solution or tighter lower bound, which lays a foundation for exploring exact algorithm for the investigated problem. 3) The improved algorithm can obtain optimal or near optimal solution for the studied problem in a short time, i.e., it is effective in both solution quality and solution time, which is verified by CPLEX. 4) The improved

algorithm is superior in terms of solution quality, computational time and robustness, thus it can be expected to provide more value and effective guidance for enterprise's logistics management

Future work will concentrate on deriving stronger valid inequalities and developing a tailored branch-and-cut algorithm to solve the problem. Moreover, time windows on pickup and delivery need to be considered for the case of emergency transshipment, as it sometimes appears in production enterprise that the adjustment of production plan results in strict arrival time of raw materials.

Acknowledgements

This research is supported by National Nature Science Foundation of China (No. 71902054 and No. 71602015), Humanities and Social Sciences Foundation of the Chinese Ministry of Education (No. 16YJC630014), and Fundamental and Frontier Research Project of Chongqing (No. cstc2016jcyjA0530).

References

1. Savelsbergh M.W.P. and Sol M., The General Pickup and Delivery Problem. Transportation Science, **29**, 17-29 (1995).
2. Toth P. and Vigo D., The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2001).
3. Chen Q., Li K. and Liu Z., Model and algorithm for an unpaired pickup and delivery vehicle routing problem with split loads. Transportation Research Part E: Logistics & Transportation Review, **69**, 218-235 (2014).
4. Xu, Dongyang, Li K., Zou X. and Ling L., An unpaired pickup and delivery vehicle routing problem with multi-visit. Transportation Research Part E: Logistics & Transportation Review, **103**, 218-247 (2017).
5. Chao L. and Yanfeng O., Continuous approximation for demand balancing in solving large-scale one-commodity pickup and delivery problems. Transportation Research Part B: Methodological, **109**, 90-109 (2018).
6. Chemla D., Meunier F. and Wolfler Calvo R., Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization, **10**, 120-146 (2013).
7. Ho S.C. and Szeto W.Y., Solving a static repositioning problem in bike-sharing systems using iterated tabu search. Transportation Research Part E: Logistics & Transportation Review, **69**, 180-198 (2014).
8. Forma I.A., Raviv T. and Tzur M., A 3-step math heuristic for the static repositioning problem in bike-sharing systems. Transportation Research Part B: Methodological, **71**, 230-247 (2015).
9. Ho S.C. and Szeto W.Y., A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. Transportation Research Part B: Methodological, **95**, 340-363 (2017).
10. Amico M.D., Iori M., Novellani S. and Subramanian A., The Bike sharing Rebalancing Problem with Stochastic Demands. Transportation Research Part B: Methodological, **118**, 362-380 (2018).
11. Maggioni F., Cagnolari M., Bertazzi L. and Wallace S.W., Stochastic optimization models for a bike-sharing problem with transshipment. European Journal of Operational Research, **276**, 272-283 (2019).
12. Hernández-Pérez H. and Salazar-González J.J., The Multi-commodity Pickup-and-

delivery Traveling Salesman Problem. Networks, **63**, 46-59 (2014).
13. Benchimol M., Benchimol P., Chappert B., Taille A., Laroche F., Meunier F. and Robinet L., Balancing the stations of a self service "bike hire" system. RAIRO - Operations Research, **45**, 37-61 (2011).
14. Erdoğan G., Laporte G. and Calvo R.W., The static bicycle relocation problem with demand intervals. European Journal of Operational Research, **238**, 451-457 (2014).
15. Szeto W.Y., Liu Y. and Ho S.C., Chemical reaction optimization for solving a static bike repositioning problem. Transportation Research Part D: Transport and Environment, **47**, 104-135 (2016).
16. Erdoğan, G., Battarra, M., Calvo, R.W., An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. European Journal of Operational Research, **245**, 667-679 (2015).
17. Pal A. and Zhang Y., Free-floating bike sharing: Solving real-life large-scale static rebalancing problems. Transportation Research Part C: Emerging Technologies, **80**, 92-116 (2017).
18. Cruz F., Subramanian A., Bruck B.P. and Iori M., A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. Computers & Operations Research, **79**, 19-33 (2017).
19. Li Y., Szeto W.Y., Long J. and Shui C.S., A multiple type bike repositioning problem. Transportation Research Part B: Methodological, **90**, 263-278 (2016).
20. Raviv T.A.T.M., Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal On Transportation and Logistics, **2**, 187-229 (2013).
21. Dell'Amico M., Hadjicostantinou E., Iori M. and Novellani S., The bike sharing rebalancing problem: mathematical formulations and benchmark instances. Omega, **45**, 7-19 (2014).
22. Kadri A.A., Kacem I. and Labadi K., A branch-and-bound algorithm for solving the static rebalancing problem in bicycle-sharing systems. Computers & Industrial Engineering, **95**, 41-52 (2016).
23. Dell'Amico M., Iori M., Novellani S. and Stützle T., A destroy and repair algorithm for the Bike sharing Rebalancing Problem. Computers & Operations Research, **71**, 149-162 (2016).
24. You P.S., Lee P. and Hsieh Y., An artificial intelligent approach to the bicycle repositioning problems. Engineering Computations, **34**, 145-163 (2017).
25. Schuijbroek J., Hampshire R.C. and van Hoeve W.J., Inventory rebalancing and vehicle routing in bike sharing systems. European Journal of Operational Research, **257**, 992-1004 (2017).
26. You P., A two-phase heuristic approach to the bike repositioning problem. Applied Mathematical Modelling, **73**, 651-667 (2019).
27. Forma I.A., Raviv T. and Tzur M., A 3-step math heuristic for the static repositioning problem in bike-sharing systems. Transportation Research Part B, **71**, 230-247 (2015).
28. Rainerharbach M., Papazek P., Raidl G.R., Hu B. and Kloimullner C., PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems. Journal of Global Optimization, **63**, 597-629 (2015).
29. Gaspero L.D., Rendl A. and Urli T., Balancing bike sharing systems with constraint programming. Constraints - An International Journal, **21**, 318-348 (2016).
30. Bulhoes T., Subramanian A., Erdogan G. and Laporte G., The static bike relocation problem with multiple vehicles and visits. European Journal of Operational Research, **264**, 508-523 (2018).
31. Hernández-Pérez H. and Salazar-González J.J., A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discrete Applied Mathematics, **145**, 126-139 (2004).
32. Hernández-Pérez H. and Salazar-González J.J., Heuristics for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem. Transportation Science, **38**, 245-255 (2004).

33. Hernández-Pérez H. and Salazar-González J., The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. Networks, **50**, 258-272 (2007).
34. Hernández-Pérez H., Rodríguez-Martín I. and Salazar-González J.J., A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. Computers & Operations Research, **36**, 1639-1645 (2009).
35. Zhao F., Li S., Sun J. and Mei D., Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. Computers & Industrial Engineering, **56**, 1642-1648 (2009).
36. Mladenović N., Urosevic D., Hanafi S.D. and Ilic A., A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. European Journal of Operational Research, **220**, 270-285 (2012).
37. Salazar-González J.J. and Santos-Hernández B., The split-demand one-commodity pickup-and-delivery travelling salesman problem. Transportation Research Part B: Methodological, **75**, 58-73 (2015).
38. Hernández-Pérez H., Rodríguez-Martín I. and Salazar-González J.J., A hybrid heuristic approach for the multi-commodity pickup-and-delivery traveling salesman problem. European Journal of Operational Research, **251**, 44-52 (2016).
39. Lu Y., Benlic U. and Wu Q., A population algorithm based on randomized tabu thresholding for the multi-commodity pickup-and-delivery traveling salesman problem. Computers & Operations Research, **101**, 285-297 (2019).
40. Zhao F., Li S., Sun J. and Mei D., Genetic Algorithm for the One-commodity Pickup-and-delivery Traveling Salesman Problem. Computers & Industrial Engineering, **56**, 1642-1648 (2009).
41. Mladenović N. and Hansen P., Variable neighborhood search. Computers & Operations Research, **24**, 1097-1100 (1997).
42. Cordeau, J.F., A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. Operation Research, **54**, 573-586 (2006).
43. Agarwal Y. and Aneja Y., Fixed-charge transportation problem: facets of the projection polyhedron. Operations Research. **60**, 638-654(2012).

MISTA 2019

Robustness Approaches for the Examination Timetabling Problem under Data Uncertainty

Bernd Bassimir · Rolf Wanka

Abstract In the literature the examination timetabling problem (ETTP) is often considered a post-enrollment problem (PE-ETTP). In the real world, universities often schedule their exams before students register using information from previous terms. A direct consequence of this approach is the uncertainty present in the resulting models. In this work we discuss several approaches available in the robust optimization literature. We consider the implications of each approach in respect to the examination timetabling problem and present how the most favorable approaches can be applied to the ETTP. Afterwards we analyze the impact of some possible implementations of the given robustness approaches on two real world instances and several random instances generated by our instance generation framework which we introduce in this work.

Keywords Examination Timetabling, Robust Optimization, Test Instance Generation

1 Introduction

In every academic term, universities are faced with a number of different aspects of academic timetabling. Academic timetabling is divided into two distinct, but similar problems, namely the *Course Timetabling Problem* (CTTP) (for a comprehensive overview, see Mühlenthaler's monograph [11]) prior to each term and the *Examination Timetabling Problem* (ETTP) at the end of the term.

CTTP is the task to assign one timeslot and one room to each lecture that is held in this term, creating a timetable that can be repeated each week over the complete term. Besides the problem of checking whether there is an admissible timetable at all,

Research funded in parts by the School of Engineering of the University of Erlangen-Nuremberg.

Bernd Bassimir (✉)
Department of Computer Science, University of Erlangen-Nueremberg, Germany
E-mail: bernd.bassimir@fau.de

Rolf Wanka
Department of Computer Science, University of Erlangen-Nueremberg, Germany
E-mail: rolf.wanka@fau.de

there is also an optimization variant, if there is a way to assess timetables and look for "good" (or best possible) timetables.

In the ETTP, one timeslot and one or more rooms must be assigned to each exam. An exam is held only once per term and as such the timetable, in contrast to the CTTP, needs not to be repeatable. At many universities, the exams are held at the end of the term in a period of a few weeks after the end of the lectures. Again, there is also an optimization variant. For a detailed model of the ETTP see [10].

Both mentioned problems are NP(O)-complete and, hence, no polynomial time (exact) algorithms are known. As each university is faced with the ETTP and the CTTP several times per year, many different approaches are proposed in the literature to solve these problems automatically.

In the literature the ETTP is often described as a problem where the number of students that are attending an exam are given as an input to the scheduling algorithm, which is sometimes called *post-enrollment* ETTP or is at least treated as a *post-enrollment* ETTP, e. g., see [6,8,12]. First the students register for their different exams and after registration is finished the optimization takes place. In this approach all parameters are known, as we have the exact number of students registered for an exam and exams are in conflict, i. e., should not be scheduled at the same time, if there are students taking both exams. In practice however some universities generate their schedules for the exams before this registrations takes place to provide a time and date for the exams when students register, e. g., see [1,4,5]. Most often the values for the number of students per exam are taken from the last year as well as the conflicts that were present in the previous year.

In this work we consider the number of students attending an exam as well as the number of students inducing a conflict for a pair of exams to be random variables. Given that these variables are not fixed as in the classical post-enrollment model we have to account for the uncertainty and consequently the deviation from the estimated values of the actual values when students eventually register for their exams. Instead of improving the estimations, which might not be possible, we focus on handling the uncertainty in the optimization process. To handle this uncertainty in the optimization process we discuss several approaches of robust optimization and their application on the ETTP. We furthermore introduce a new instance generation framework, from which we obtain a more diverse test set.

In Sect. 2 we introduce the basic model used in this work and the implications of the uncertainty on this model. In Sect. 3 we introduce several different strategies for robust and stochastic optimization to deal with this uncertainty and in Sect. 4 we introduce a framework for the generation of random instances and the simulated annealing algorithm we used in our experiments. In Sect. 5 we compared implementations of two robustness approaches on two real world instances taken from the University of Erlangen-Nürnberg and random instances generated with the newly introduced instance generation framework.

2 Model

In [10], McCollum et al. introduce an extensive model for the Examination Timetabling Problem. For the purpose of this paper we use a reduced version of this model with only a subset of the soft constraints.

Definition 1 *An instance of the Examination Timetabling Problem is represented as follows.*

- $\mathcal{E}$*: A set of exams*
- $\mathcal{R}$*: A set of rooms*
- $\kappa : \mathcal{R} \to \mathbb{N}$*: the capacity available for each room and* $\kappa : \mathcal{P}(\mathcal{R}) \to \mathbb{N}$ *as the canonical extension such that for* $X \subseteq \mathcal{R} : \kappa(X) := \sum_{r \in X} \kappa(r)$
- $\nu \in \mathbb{N}^{|\mathcal{E}|}$*: A vector specifying for each exam how many students are attending*
- *A conflict matrix* $C \in \mathbb{N}^{|\mathcal{E}| \times |\mathcal{E}|}$ *specifying the number of overlapping students of two exams*
- *A set of soft constraints with associated weights*

Hard Constraints To be feasible, a timetable must meet the following hard constraints:

(H1) each exam is assigned to exactly one timeslot
(H2) each exam is assigned to one or more rooms
(H3) two exams that are in conflict are not scheduled at the same time
(H4) no room is used at the same time by two exams
(H5) the sum of capacities of the assigned rooms is larger than the number of students taking the exam

Most of this model is similar to the model in [10] except that we allow multiple rooms per exam.

Soft Constraints In our reduced model we use the soft constraints *two-in-a-row*, *two-in-a-day* and *period-spread* as defined in [10].

(S1) *two-in-a-row*: If two exams are in conflict according to C they should not be assigned to two adjacent timeslots on the same day.
(S2) *two-in-a-day*: If two exams are in conflict according to C they should not be assigned to two timeslots on the same day. Note that we exclude the directly adjacent timeslot to avoid double counting.
(S3) *period-spread*: If two exams are in conflict according to C they should not be assigned to timeslots less than λ apart.

Each violation of a soft criterion induces a penalty corresponding to the number of students that are in conflict given by C. Given a feasible timetable, i. e., a timetable that satisfies hard constraints (H1) - (H5), we can formulate the objective value as the weighted sum of the penalties induced by the soft constraints (S1) - (S3). As a violation of soft constraint (S1) infers a violation of soft constraint (S2) we exclude the adjacent timeslot in the calculation of the violations of soft criterion (S2). For the soft criterion (S3) we ignore overlaps and compensate this in the weight. The weights of the penalties can vary for each university and therefore are not fixed in the model. For a possible selection see Sect. 5.

2.1 Examination timetabling under uncertainty

In real world examination timetabling the schedule is sometimes generated before students register for exams. This approach can be called Curriculum-Based analogue to the course timetabling problem. In such a situation the exact number of students attending an exam and the number of students inducing conflicts for exams are not

known at the time of optimization. However usually we have static information available provided by the curriculum model. There is the curriculum data Curr: $\mathcal{E} \to \mathcal{P}(\mathcal{M})$ assigning each exam a set of major and semester combinations $\mathcal{M}$, such that students in this combination of major and semester can attend the respective exam and the registration data Enroll: $\mathcal{M} \to \mathbb{N}$ specifying the number of students registered for each major and semester combination.

We treat the number of students attending an exam e as a random variable ν_e with an unknown discrete distribution on the support $\{1, 2, \ldots, \sum_{c \in \mathrm{Curr}(e)} \mathrm{Enroll}(c)\}$. Note that we exclude 0 as this would imply that nobody would take the exam and therefore the exam would not be scheduled. We ignore this case as all exams have to be scheduled. Furthermore as there are optional courses we can not quantify the number of students that induce a conflict between courses. We therefore treat the weight of a conflict $C_{i,j}$ as a random variable with a discrete support on $\{0, 1, \ldots, \min\{\nu_i, \nu_j\}\}$. Here 0 is a valid value and specifies that the conflict is not present. We call the vector ν and the matrix C the uncertain parameter of our model. We call the set of possible realizations of the parameters the uncertainty set $\mathcal{U}$ [2, Def. 1.2.1]. Given this uncertainty in our model we have to revisit the hard constraints (H3) and (H5) as they depend on these random values.

3 Robustness Measures

As previously discussed, there are two possibilities for the time when the scheduling is performed. The first variant is to start the scheduling process after students have registered for all exams they want to take. At that time the value for each random variable is known, which gives us a model without uncertain parameters that can be solved conventionally. The other variant, which is the focus in this work, is to schedule the exams before registration takes place. This presents us with challenges regarding several aspects of the scheduling process. First and foremost we do not know how many students attend an exam. Furthermore we do not know which exams are actually in conflict.

As introduced in Section 2.1 the curriculum information provides us with upper bounds on the number of students attending an exam and the data for which exams can be in conflict as well as the upper bound on the number of students inducing the conflict. However the range for the uncertain parameters is quite large and therefore using their upper bounds is problematic as this can and most of the time will lead to infeasible problems.

For optimization problems under uncertainty we can use the methods of robust or stochastic optimization to find a solution that is feasible for all or at least most realizations of the uncertain data. In this section we will give a brief overview of common robust and stochastic optimization techniques for handling uncertainty in the problem model. For an extended discussion on these topics we refer to Ben-Tal et al. [2] for robust optimization and Birge and Louveaux [3] for stochastic optimization.

The basic optimization problem we want to optimize can be defined as a general linear program.

$$\begin{aligned} \min_{x} \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{1}$$

For a detailed description on how the examination timetabling problem can be expressed as a linear program see [10]. In our case the weights c, the constraint matrix A and the vector b are subject to uncertainty. However the uncertainty in c is ignored and we focus on the uncertainty in A and b as these can lead to violations of constraints in contrast to the impact of c being local to the objective value. Each uncertain variable ζ in the uncertainty set $\mathcal{U}$ can be seen as $\zeta = \bar{\zeta} + \delta$, where $\bar{\zeta}$ is the nominal value, i. e., the value if no deviation from the expected scenario is present, and the disturbance δ representing the possible deviations from the expected scenario. Please note that δ can be negative. A timetable is called nominal feasible if there is no violation of the hard constraints when using the nominal values for each uncertain parameter. We further call a timeslot feasible if there is no violation of the hard constraints present for the exams in this timeslot otherwise the timeslot is called *disturbed*. Each timeslot contains a subset of the exams $\mathcal{E}$ with an assignment of rooms for each exam. We call such an assignment of rooms for an exam e room pattern $RP(e)$ with the corresponding capacity $\kappa(RP(e)) = \sum_{r \in RP(e)} \kappa(r)$.

We will use this general model to introduce the different robustness approaches and will discuss the implications of the uncertain examination timetabling problem for the different approaches.

3.1 Strict Robustness

The first and simplest approach for robust optimization is called strict robustness. A solution is called strict robust, iff it is feasible for all realizations of the uncertain data. The tractability of this approach is in large parts dependent on the uncertainty set. In the case of examination timetabling we have an interval uncertainty set $\mathcal{U}$. For interval uncertainty we get a linear program with only linear overhead in the number of constraints. In our particular case we can just use the upper bound on all uncertain parameters. A solution for the resulting linear program will be feasible for all possible scenarios of the uncertain data. The existence of a feasible solution for the linear program given our uncertainty set $\mathcal{U}$ based on curriculum data is highly unlikely due to the far too pessimistic upper bounds on the number of students and the pessimistic assumption that all possible conflicts are present. However with more realistic bounds this approach could get viable, but better information is usually not available.

3.2 Stochastic Optimization

Another approach to handle uncertainty in the model is stochastic optimization, see [3]. In contrast to robust optimization, that tries to find the best feasible solution for all possible scenarios of the uncertain parameters, stochastic optimization uses stochastic measures to optimize the model.

Two-stage Stochastic Optimization In the two-stage approach we categorize the variables into two distinct groups. The first group are the primary, *first-stage* variables. These variables have to be set before the scenario is realized. The second group, the *second-stage* variables however can be set after the scenario is known. In this setting the linear program transforms to

$$\begin{aligned} \min_{x} \quad & c^T x + E[\min_{y} \; q^T y] \\ s.t. \quad & Ax = b \\ & Tx + Wy = h \\ & x \geq 0 \quad . \end{aligned} \tag{2}$$

In this model the variables x are the *first-stage* and the variables y the *second-stage* variables. The goal of the optimization is to find an optimal assignment of the *first-stage* variables, such that in expectation the objective depending on the *second-stage* variables is minimal.

In the case of examination timetabling a possible division of the variables is to set the variables specifying the timeslot of an exam as *first-stage* and the variables specifying the rooms as *second-stage* variables. We however do not have the exact distributions for our random variables and they might change over time. Given this problem there are two approaches to still use the technique of stochastic optimization. The first is to use some kind of fitted distribution, e. g., normal distribution in the optimization process in the hope that the actual numbers do not deviate too heavily from the fitted distribution. The other approach is to calculate an empirical distribution from the previous years. As with a fitted distribution the usefulness depends heavily on the deviations of the actual scenario.

3.3 Recoverable Robustness

In some cases we might have the situation that the error situation is quite unlikely. In such a case using stochastic optimization or especially strict robust optimization might be too conservative. In this section we introduce the principle of recoverable robustness. In recoverable robustness the scheduling uses the nominal values for all uncertain parameters. Therefore the returned solution is allowed to become infeasible when the scenario is realized. However for each solution to be feasible in the recovery robust model there needs to exist an algorithm to recover from the infeasibility.

In theory recovery can become quite expensive in regard to the objective value therefore in practice the cost of the recovery is often bounded. These bounds come in two variants, namely with a fixed budget or as an addition to the objective value to be minimized.

Given the linear program with the nominal variables $\bar{A}$ and $\bar{b}$, i. e., the values if no errors are present compared to the estimation, and the recovery matrix $\hat{A}$, i. e., the influence of the recovery vector y in the constraints, we get for a fixed budget D the

problem

$$\begin{aligned} \min_{x} \quad & c^T x \\ s.t. \quad & \bar{A}x \leq \bar{b} \\ & \forall (A,b) \in \mathcal{U} \; \exists y : Ax + \hat{A}y \leq b \\ & d^T y \leq D \\ & x \geq 0 \end{aligned} \tag{3}$$

and the problem for a flexible budget λ to be minimized

$$\begin{aligned} \min_{x,\lambda} \quad & c^T x + t\lambda \\ s.t. \quad & \bar{A}x \leq \bar{b} \\ & \forall (A,b) \in \mathcal{U} \; \exists y : Ax + \hat{A}y \leq b \\ & d^T y \leq \lambda \\ & x \geq 0 \quad . \end{aligned} \tag{4}$$

In academic timetabling it is possible, due to a change in the staff or similar occasions, that a released timetable might become infeasible, i. e., a lecturer might change to another university and the replacement lecturer can not lecture the course or exam in the assigned timeslot. In this case the afflicted course or exam needs to be rescheduled, however this rescheduling should not influence the whole timetable but be mostly local, e. g., should not affect more than two timeslots.

In the case of data uncertainty of the number of students taking an exam we can not limit our robustness approach to single disturbances only. When using estimations for these numbers as nominal values in reality we can expect to have a deviation from these values for each exam. As such there can be several exams that have an insufficient room assignment and our optimization approach needs to handle all possible sets of these infeasible exams.

Our approach In our robust optimization approach on recovery we distinguish between two types of variables similar to the two-stage stochastic optimization. We consider the variables in our model specifying the timeslot of an exam as *first-stage* and the assignment of rooms to exams as *second-stage* variables. The reassignment of the rooms of a timeslot are therefore the y variables in Eq. (3) and (4) with a cost vector $\mathbf{0}$. Only the timeslot has a direct impact on the quality of the solution. This enables a potential recovery without an increase in the quality of the solution. We first optimize the given instance using an estimation for the different uncertain parameter. After the actual registration of students for exams the scenario is realized, which in turn can lead to timeslots that are infeasible in respect to the actual values in the scenario. There are three types of exams in such a timeslot. Exams with more, equal to or less actual students than anticipated. Furthermore as we do not enforce the use of all rooms in a timeslot we might have unassigned rooms available.

In the following we present two recovery strategies that are local to the timeslot and therefore will not affect the solution quality.

The goal of the *Heuristic Recovery* strategy in Algorithm 1 is to find a room assignment for the infeasible timeslot without negatively affecting the feasible exams.

Algorithm 1: Heuristic Recovery

input : Infeasible Timeslot t
output: Feasible Assignment or notice of failure
for *exam* e **in** t, $\phi(RP(e))) \geq \nu_e$ **do**
    Reduce rooms of exam e in t, such that the number of rooms is minimal
Assign free rooms to all exams e in t with $\phi(RP(e)) < \nu_e$
if *feasible assignment of rooms for all exams in* t *exists* **then**
    return assignment
else
    return failure

If an exam has the same or fewer students than in the nominal model, the recovery is only allowed to remove rooms but not to add any. The strategy starts with removing all unnecessary rooms from feasible exams. Afterwards all now unassigned rooms are assigned to the infeasible exams. If such an assignment exists it is returned otherwise the timeslot is marked as infeasible. The advantage of this strategy is that it only removes rooms from the exams that have an feasible assignment and therefore the organizers of such an exam do not have to increase their assigned staff for the exam.

Algorithm 2: Complete Recovery

input : Infeasible Timeslot t
output: Feasible Assignment or notice of failure
Remove all rooms from the exams in t
Reassign all rooms to the exams in t

The *Complete Recovery* strategy in Algorithm 2, while still having no impact on the objective value changes the room assignment completely. In the first stage all room assignments are removed and afterwards a new feasible assignment is computed for this timeslot. For the organizers of an exam this means that they might need more persons supervising the exam than originally planned even if their exam had a satisfying room assignment. However if there is a possibility to recover from the disturbances this strategy can find it. Both strategies do not necessarily produce feasible timetables. If the strategy can not recover all infeasible timeslots other recovery strategies have to be used.

3.4 Light Robustness

Light robustness is an approach that incorporates aspects of stochastic and recoverable robustness, which was introduced by Fischetti and Monaci [7]. The basic idea introduced by Fischetti and Monaci is to first calculate the nominal optimal solution and afterwards to find the most robust solution, where objective values deviate only by a fixed amount.

Furthermore they introduce a heuristic approach for light robustness. As before a nominal optimal solution is first calculated. Afterwards a slack variable is introduced for each uncertain constraint. Using these slack variables the model is solved again with an upper bound on the difference to the nominal optimum. In this step the new

objective is to maximize the minimum of the normalized slack variables. Unlike strict robustness the model does not guarantee the feasibility in all scenarios of the uncertain data, but tries to find the best solution in regards to a robustness measure within a given range around the nominal optimum. In a third optimization step a solution is calculated that balances the normalized slack variables while the objective value is only allowed to deviate by a fixed value. In this step the minimum of the normalized slack variables as calculated in the second optimization step is used as a lower bound for the normalized slack variables.

In [1] we introduced an approach called probabilistic curriculum-based examination timetabling (PCBETT) that is quite similar to the heuristic approach of Fischetti and Monaci. The main difference is that instead of first calculating the nominal optimum and afterwards optimizing a robustness measure we incorporate the robustness measure in regard to the number of students per exam directly into the optimization. For this purpose we introduced the new soft criterion (5) in [1].

$$\max \left\{ \frac{\nu(e)}{\kappa(RP(e))} \mid e \in \mathcal{E} \text{ , } RP(e) \text{ is room pattern of } e \text{ in the current timetable} \right\} \quad (5)$$

with $\kappa(RP(e)) = \sum_{r \in RP(e)} \kappa(r)$ the canonical extension as introduced in the model of Sect. 2. This new soft criterion should be minimized. As this introduced soft criterion is non-linear we use a slight but semantically equivalent reformulation for this paper that can be linearized by standard transformations.

$$\text{(S4)} \quad \min \left\{ \frac{\kappa(RP(e))}{\nu(e)} \mid e \in \mathcal{E} \text{ , } RP(e) \text{ is room pattern of } e \text{ in current timetable} \right\}$$

This reformulated soft criterion has to be maximized. In contrast to the approach introduced by Fischetti and Monaci we use the soft criterion directly in our optimization and only solve the problem once as instances for the ETTP can be quite large. Note that this approach does not handle the uncertainty regarding conflicts.

4 Instance Generation and Optimization

In Sect. 3 we introduced several strategies to get a robustness aspect into the examination timetabling optimization. To get an understanding of the performance of these different strategies we compare them on two real world instances from the University of Erlangen-Nürnberg. However to increase our test size we introduce an instance generation algorithm in Sect. 4.1 to generate random instances with different characteristics in respect to structure of the conflicts. These instances were than optimized by a simulated annealing algorithm described in Sect. 4.2.

4.1 Instance Generation

In optimization it is often interesting to study real world instances and the influence of different approaches on them. However to get a deeper understanding of these techniques and to rule out unique influences present in our real world instances we have to increase and diversify the test data. There are two approaches to accomplish this diversity. The first and often used is the collection of different instances from other universities. In the case of the characteristics of our model, e. g., the assignment of

multiple rooms to an exam, there exists to our knowledge no such collection. Furthermore it would still be limited to only a small number of different instances. To have a diverse set of instances, where we can produce different characteristics and study their influence on the performance of the robustness approaches we introduce a random instance generation framework.

Algorithm 3: Instance Generation

input : Number of exams E, Number of rooms R, Number of timeslots T
output: ETTP Instance

$\mathcal{R}$:= set of rooms of size R
for r **in** $\mathcal{R}$ **do**
  /* $v \sim \mathrm{NB}(0.2, 25)$ in our tests, other distributions are possible */
  choose $v \sim \mathrm{NB}(0.2, 25)$
  set $\kappa(r) := v$
$\mathcal{E}$:= set of exams of size E
Generate random partition $EP = (EP_1, EP_2, \ldots, EP_T)$ of $\mathcal{E}$ into T sets each of size between 1 and R
for t **from** 1 **to** T **do**
  Generate random assignment RP of rooms in $\mathcal{R}$ to exams in EP_t, which is a partition of $\mathcal{R}$ with sets of size at least 1
  for e **in** EP_t **do**
    /* $n \sim \mathrm{Beta}(10, 5)$ in our tests, other distributions are possible */
    choose $n \sim \mathrm{Beta}(10, 5)$
    $\nu(e) := 1 + \lfloor \kappa(RP(e)) \cdot n \rfloor$
for i **from** 1 **to** E **do**
  for j **from** 1 **to** E, $i \neq j$ **do**
    choose u u.a.r. from $[0, 1]$
    /* $c(i, j)$ is the predefined probability for exams i and j to be in conflict */
    if $u < c(i, j)$ **then**
      $C_{i,j} := 1$
    else
      $C_{i,j} := 0$

Algorithm 3 shows the framework for our instance generation which is highly flexible as several different distributions that can be used for the random variables. We start with generating a number of rooms. For our tests room sizes are sampled according to a negative binomial distribution with the parameter $p = 0.2$ and $k = 25$. This gives us a selection of rooms with mostly medium to small rooms and only a few large rooms. Afterwards we generate a random partition of the exams into T sets, where T is the number of timeslots and each set of the partition contains between 1 and up to $|\mathcal{R}|$ exams. For each timeslot we generate a random assignment of rooms to the exams in the timeslot, where a room is assigned to exactly one exam. The number of students is then drawn at random between 1 and the assigned room capacity $\phi(RP(e))$. Using this generation process we can ensure that without conflicts the instance is feasible. Afterwards we generate conflicts for the exams by predefined probabilities for each conflict. In our experiments we used two different types of choices for the probability of the conflicts, see Sect. 5.

4.2 Optimization Framework

As our models can get quite large and therefore can most often not be solved to optimality we use a heuristic approach to solve the examination timetabling problem. To perform the optimization of our two real world instances and the instances generated by our instance generation framework we implemented a simulated annealing algorithm. The used algorithm is quite similar to the simulated annealing algorithm described by Mühlenthaler in [11]. It uses the Kempe-Exchange neighborhood to select a new neighbor.

The differences between the simulated annealing algorithm as introduced in [11] and the algorithm used for our optimization are the feasibility of the room assignment and the cooling scheme.

A neighbor is only considered if there exists a room assignment for each of the two changed timeslots. We used Cplex [9] to calculate the room assignments for each timeslot. This is necessary to ensure the feasibility of each consecutive step in the annealing phase. For the nominal solution without the use of a robustness measure we just return the first solution found by the solver. When using the PCBETT approach the room assignment is instead optimized using only soft constraint (S4) as the objective and the optimal solution of the room assignment in regards to (S4) is returned. As only one timeslot is considered in this optimization the feasibility of the timeslot depends only on the hard constraint (H5) which is satisfied by using soft constraint (S4) if a feasible solution exists.

Experiments performed using this simulated annealing algorithm showed a fast convergence to a local minimum with only a small chance to leave this local optimum. To counter this phenomenon we modified the cooling scheme. After a predefined amount of iterations we reset the heat level. Furthermore to counter a strong drift while not in a local optimum we doubled the cooling rate if a better solution was accepted in the iteration.

Using this cooling scheme we could increase the performance of the simulated annealing algorithm, however the problem could not be avoided. To further improve the returned solution we started the simulated annealing algorithm several times but with fewer iterations. The best solution found in the different runs was then returned, which could significantly improve the performance of the overall algorithm.

As a starting point we used a simple heuristic that schedules the exams in decreasing order in respect to the number of students taking the exam. Rooms are assigned in decreasing order in respect to the capacity of the room until the assigned capacity is larger than the number of students taking the exam. If not enough rooms are available in the timeslot the next timeslot is used. In practice the heuristic proved to be sufficient for an initial feasible solution. The objective value however was quite far from optimal, but can be optimized by the simulated annealing quite drastically.

5 Robustness Evaluation

To compare the different robustness approaches introduced in Sect 3 we used two real world instances from the University of Erlangen-Nürnberg and the instance generation framework introduced in Sect. 4.1. With this framework we generated 20 instances for the 5 different instance generation settings as shown in Table 1 to have a better

Scenario	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	T	ν_e	κ	Conflict generation
Scenario 1	120	8	30	Beta(10, 5)	NB(0.2, 25)	chosen u.a.r. with $p = 0.1$
Scenario 2	120	8	30	Beta(10, 5)	NB(0.2, 25)	chosen u.a.r. with $p = 0.25$
Scenario 3	90	8	30	Beta(10, 5)	NB(0.2, 25)	Poisson-binomial with $p_e \sim$ Beta(4, 6)
Scenario 4	120	8	30	Beta(10, 5)	NB(0.2, 25)	Poisson-binomial with $p_e \sim$ Beta(4, 6)
Scenario 5	150	8	30	Beta(10, 5)	NB(0.2, 25)	Poisson-binomial with $p_e \sim$ Beta(4, 6)
Summer 2018	405	13	75	Summer term 2018		
Winter 2017	411	13	75	Winter term 2017		

Table 1 Instance generation settings used in the evaluation of the robustness approaches

understanding of the approaches for different characteristics in examination timetabling instances.

In our test sets we used two different types of choices for the probability of a conflict. The first and most simple conflict generation generates a conflict between two exams uniform at random (u.a.r.) with a given probability. As such exams have around the same number of conflicts each. This however is quite far from an actual distribution of conflicts in real word data. To increase the similarity between the randomly generated instances and actual real word instances we generated the conflicts Poisson-binomial distributed. For each exam we draw a conflict probability according to a Beta(4, 6) distribution. This beta distribution creates the situation that most exams have a medium to low number of conflicts with only a few exams having a high number of conflicts. The probability of a conflict is then calculated as the product of the exam probabilities of the exams in the conflict. This results in a Poisson-binomial distribution for the conflicts. The number of exams and timeslots are chosen to have a ratio similar to our real world instances.

As described in [1] for our real world instances we used the information taken from the previous years for the number of students taking an exam and the number of students inducing a conflict is the arithmetic mean taken over the previous years.

Each generated instance is then optimized by the simulated annealing algorithm introduced in Sect. 4.2 with parameters $t \in [1, 100]$ as heat level, a cooling limit of 5 000, 50 000 iterations and 8 repetitions. We used the soft constraint weights 300 for (S1), 150 for (S2) and 5 for (S3) with a period spread of $\lambda = 4$. Each instance is optimized twice, once with the soft constraint (S4) and weight 3000 and once without it. We then take the arithmetic mean over the 20 different instances for each soft criterion. As already a single optimization run of a single instance has a runtime up to an hour it is not possible to increase the test size significantly and still evaluate all tests in a reasonable amount of time. For the two real world data instances we increased the iterations to 150 000 and the cooling limit to 50 000.

The results for the experiments are presented in Table 2 and the weighted sum of the objective values is additionally visualized in Fig. 1. Note that the weighted sum is only calculated over the soft criteria (S1) to (S3) as the nominal solution does not use soft criterion (S4). The value of (S4) in the nominal solutions is therefore very close to 1. Through the addition of the soft criterion (S4) the objective values of the soft criteria (S1) to (S3) tend to be worse, however the difference is quite low.

The second test uses the instances and timetables generated by the first test and introduces errors into the timetables. As argued in Sect. 2.1 each exam is subject to errors. The new value for the number of students for exam e is chosen as $\nu_{e,new} \sim N(\nu_{e,old}, \frac{\nu_{e,old}}{5})$. For each instance this experiment is repeated 100 times and

	Nominal				PCBETT				
Scenario	Objective	(S1)	(S2)	(S3)	Objective	(S1)	(S2)	(S3)	(S4)
Scenario 1	15	0	0	3	294	0	0	58.9	1.62114
Scenario 2	38649.75	2.9	57.75	5823.45	39645.25	2.8	62.95	5872.55	1.08674
Scenario 3	56.25	0	0	11.25	205.75	0	0	41.15	1.62793
Scenario 4	7383.25	0.2	1.55	1418.15	8397	0.15	2	1610.4	1.27832
Scenario 5	24671.5	1.45	28.8	3983.3	25247.5	1.6	26.5	4158.5	1.12212
Summer 2018	8645	0	33	739	9170	0	38	694	2.21794
Winter 2017	13925	0	57	1075	14255	2	53	1141	1.92650

Table 2 Influence of the pcbett approach on the objective function. Each value is the arithmetic mean over 20 generated instances generated per scenario as well as the objective function values for the two real world data instances.

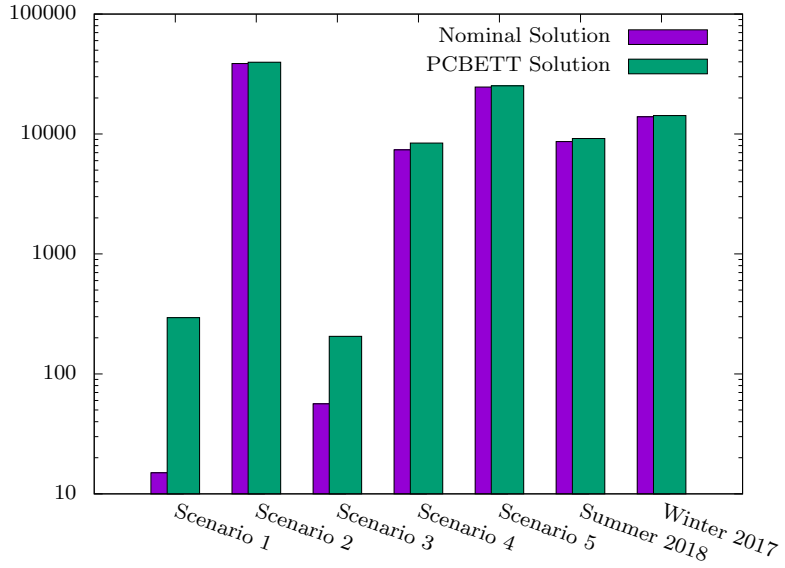


Fig. 1 Arithmetic mean of the weighted objective values calculated over 20 generated instances per scenario and the weighted objective values of the two real world data instances.

the timetables are evaluated with the new values ν_e. For each timetable the number of infeasible timeslots are counted and the arithmetic mean is calculated.

Table 3 shows the results for the arithmetic mean over the 20 test instances and the 100 disturbances. We can see, that the number of disturbed timeslots when using the soft criterion (S4) is significantly lower than the number of disturbed timeslots for the unmodified nominal solution. It is also noticeable that the repair heuristic and especially the complete recalculation can significantly reduce the number of timeslots that are infeasible. For the heuristic the number is however still bigger than the unmodified optimization results with soft criterion (S4). If we allow the complete recalculation of the disturbed timeslots the actual infeasible timeslots can be reduced significantly while still no change in the objective function occurs. However as discussed in Sect. 3.3 this recalculation can change the assignment of rooms quite drastically and therefore have negative impact on the acceptance of the teaching staff.

	Nominal			PCBETT		
Scenario	Unmodified	Heuristic	Complete	unmodified	Heuristic	Complete
Scenario 1	8.05500	4.46500	1.34250	0.03150	0.01750	0.00100
Scenario 2	8.57250	5.44850	1.85000	4.01350	3.46500	1.49700
Scenario 3	6.57750	2.84650	1.02100	0.16700	0.09750	0.03100
Scenario 4	8.29000	5.19900	2.00100	1.42950	1.07250	0.33550
Scenario 5	10.53950	7.43650	2.30950	4.03200	3.51250	1.38050
Summer 2018	13.18000	3.13000	0.65000	0	0	0
Winter 2017	12.58000	2.69000	0.16000	0	0	0

Table 3 Number of disturbed timeslots before and after robust recovery for the nominal and the pcbett solutions.

6 Conclusion

In this work we introduced several approaches of robust and stochastic optimization and discussed their applicability for the examination timetabling problem. We furthermore introduced a new random instance generation algorithm, which allowed the generation of an increased number of test instances with different characteristics. Given the quite large uncertainty sets resulting from curriculum information we discussed the problems for the different robustness approaches. We introduced two possible recovery algorithms that do not change the function value as they only consider one timeslot. The experimental results showed that especially the complete recovery algorithm was able to repair most of the disturbed timeslots however at the cost of the complete reassignment of rooms which might have a negative impact on the acceptance from the staff holding the exams. As a light robustness approach we could experimentally show that the introduction of another soft constraint was able to significantly reduce the number of timeslots that get disturbed, i. e., have exams with an insufficient room assignment, with only a small impact on the overall quality of the solution.

References

1. Bernd Bassimir and Rolf Wanka. Probabilistic Curriculum-based Examination Timetabling. In *Proc 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 273–285, 2018.
2. A. Ben-Tal, L.E. Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, 2009.
3. John R. Birge and Franois Louveaux. *Introduction to Stochastic Programming*. Springer Publishing Company, Incorporated, 2nd edition, 2011.
4. Alejandro Cataldo, Juan-Carlos Ferrer, Jaime Miranda, Pablo A. Rey, and Antoine Sauré. An integer programming approach to curriculum-based examination timetabling. *Annals of Operations Research*, 258(2):369–393, 2017.
5. Peter Demeester, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling*, 15(1):83–103, 2012.
6. Michael Eley. Ant algorithms for the exam timetabling problem. In *Pro. 6th Int. Conf. on Practice and Theory of Automated Timetabling (PATAT)*, pages 364–382, 2007.
7. Matteo Fischetti and Michele Monaci. *Light Robustness*, pages 61–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
8. Christos Gogos, Panayiotis Alefragis, and Efthymios Housos. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals of Operations Research*, 194(1):203–221, 2012.
9. IBM Corporation, Armonk, New York. *CPLEX Interactive Optimizer version 12.6.0.0*, 2013.

10. Barry McCollum, Paul McMullan, Andrew J. Parkes, Edmund K. Burke, and Rong Qu. A new model for automated examination timetabling. *Annals of Operations Research*, 194(1):291–315, Apr 2012.
11. Moritz Mühlenthaler. *Fairness in Academic Course Timetabling*. Springer, 2015.
12. Tomáš Müller. ITC2007 solver description: a hybrid approach. *Annals of Operations Research*, 172:429–446, 2009.

MISTA 2019

A computational experimentation for the flow shop scheduling problem with due date assignment

Helio Yochihiro Fuchigami

Abstract This research addresses the flow shop with due date assignment problem that can be found in several practical applications: manufacturing shops, maintenance activities, wafer fabrication, make-to-order production-distribution logistic system, potential disruptive environment, real-life situations where due date is determined during sales negotiations with the customer. It was identified the literature gap where few of papers about due date assignment dealt with flow shop problem and none of them had proposed mathematical models with computational experiences. In this work, four different strategies were employed, including position-based and sequence-based variables. An expressive computation experimentation was conducted with a modern modelling language, the Julia/JuMP. Numerical results with a well-known instances benchmark of the literature with majority large problems indicate the position-based strategies are more efficient in terms of optimality gap and CPU time consuming than the sequence-based ones.

1 Introduction

In the area of just-in-time scheduling, the objective is generally to minimize a cost function associated with both early and tardy completion of jobs. In practice it is known that earliness and tardiness of jobs should be avoided because both entail relatively higher production costs, such as increases in inventory levels, material movement and perishability, fines, cancellations of orders or even loss of customers. A broad theoretical framework and applications about just-in-time scheduling problems can be found in Ríos-Solís and Ríos-Mercado (2012) and Józefowska (2007).

Although it is very understood the importance of meeting due dates in actual practice, a major thrust of scheduling research has been directed towards improving performance with respect to steady due dates. Typically, the due dates have been assumed to be an *exogenous* variable, outside the control of the scheduler (De et al., 1992; Kaminsky and Lee, 2008). It seems to be Conway (1965) the first to introduce the concept of an attainable due date, which is determined *endogenously* by the scheduler based on workload and other characteristics of the scheduling environment. Mor (2018) stated the scheduler need to balance the trade-off between short and competitive due dates and late and unchallenging due dates.

The overall goal of the companies is to meet the proposed due dates, which depend not only on the scheduling procedure followed but also on the "reasonableness" of the due dates, defined as a measure of the due date performance reflected on the capability of the system to

Helio Yochihiro Fuchigami
Federal University of Goias (UFG), Faculty of Sciences and Technology (FCT), Brazil.
E-mail: heliofuchigami@ufg.br

achieve successfully an arbitrary set of due dates. There are two aspects of due date performance: delivery reliability which is the ability to consistently meet promised due dates given the critical importance of the fulfilment of the promised due dates; and delivery speed, which is the ability to deliver orders to the customers with shortest lead times (Perez-Gonzalez and Framinan, 2010).

About the business challenges, as pointed by Birman and Mosheiov (2004), a careful decision of the due dates in the course of sales negotiations with a customer is clearly an important task for the company. Quaddus (1987) already noted the due dates or delivery times play an important role in job sequencing problems, demanding to assign attainable due dates to the jobs. And according to Vallada et al. (2008), in industrial plants, criteria based on due dates for delivery have become more important than those based on makespan (the maximum completion time of all jobs).

The *due date assignment problem* can be found in several practical applications: lightguide cable manufacturing shop (Rajasekera et al., 1991), maintenance activity (Mosheiov and Oro, 2006), semiconductor manufacturing of wafer fabrication (Chen, 2013), make-to-order production-distribution logistic system (Jing, 2013), potential disruptive environment arising from worker or power shortage etc. (Xiong et al., 2018). Motivated by the very frequent real-life situation where the due date is determined during sales negotiations with the customer, this research addresses the due date assignment flow shop problem, which is a popular and still challenging topic of study.

According to the known three-field notation suggested by Graham et al. (1979), the problem considered in this paper can be denoted by $F_m|prmu, d|ETd$, where F_m indicates the m-machine flow shop environment, *prmu* and *d* denote respectively the restrictive requirements of permutation schedule and common due date for all jobs, and *ETd* is the performance criterion compounded by the costs of earliness, tardiness and the due date.

In this class of problems, usually three types of costs are considered: (1) earliness cost, represented by the cost of finished goods inventory; (2) tardiness cost or cost of missing a due date assumed to be a fixed cost and different for each job that can be a contractual penalty or a chance-weighted estimate of lost sales which may result from non-compliance with a quoted due date; and (3) due date costs or cost of quoting long due dates, essentially opportunity costs, assumed to become effective beyond an acceptable limit and through of as the estimated cost of either an incentive needed to induce a customer to accept a long due date quotation or the potential loss of sales associated with such a quotation (De et al., 1992).

Despite several papers on due date assignments are available in the literature, as can be seen in the surveys of Gordon et al. (2002a), Perez-Gonzalez et al. (2009) and Janiak et al. (2015), few of them dealt with the flow shop problem and according to our knowledge none of them have proposed mathematical models with computational experiences to solve the problem. That is the reason we have chosen this solution approach to find the optimal solution in this research.

2 Literature review

The class of due date assignment problems is a challenging topic where the due date itself is a decision variable, in contrast to the general case in which it is a given parameter of the problem instance. Comprehensive surveys about scheduling problems related with due dates can be seen in Sen and Gupta (1984), Baker and Scudder (1990), Hall et al. (1991), Koulamas (1994), Lauff and Werner (2004), Keskinocak and Tayur (2004), Perez-Gonzalez et al. (2009) and Shabtay and Steiner (2012).

The pioneer on due date assignment Conway (1965) considered different policies to determine due dates for the job shop problem with mean tardiness minimization and tested the performance of nine priority dispatching rules. Under the known EDD (earliest due date) rule, the policies that take the work content into account perform better. Specifically on due date assignments problems, the state-of-the-art for the moment were reviewed by Ragatz and Mabert (1984), Cheng and Gupta (1989), Gordon et al. (2002a, 2002b), Gordon et al. (2004)

and Kaminsky and Hochbaum (2004). Janiak et al. (2015) published a survey on scheduling problems with due windows, a generalization of due date assignment where jobs completed prior to or after the window are penalized.

In this context, it could arise que question: why a common due date is assigned to the jobs as a specific deadline for each job could be more accurate? However this feature depends on the environment to be studied. As pointed out by Gordon et al. (2002a), the common due date model emerges for instance to an assembly system in which the components of the product should be ready at the same time, or to a shop where several jobs constitute a single customer's order. Mohamad and Said (2011) also exemplifies the case a firm that install a weekly bulk delivery to the wholesaler.

Based on the literature analysis, in addition to the more frequent situation of individual and steady due dates for each job and the due window described above, it can be seen three other cases: (1) jobs with a common due date, such as in Alvarez-Valdes et al. (2012) and Perez-Gonzales and Framinan (2015); (2) jobs with due date constraint (as hard constraint), i.e., no delay is allowed and there no feasible solution with late jobs, as treated by Samaghandi (2015) and Samaghandi and Behroozi (2017); (3) common due date assignment for jobs, which is the case of the present research, the only one that treat the due date as a variable of the problem.

As pointed out by Keskinocak and Tayur (2004), most of the research in scheduling either ignores due dates or assumes that they are set exogenously (e.g., by the customer) and many papers assume the due dates are set internally by the firm. The reality is in between these two extremes: in most business-to-business transactions, due dates are set through a negotiation process between the customer and the firm.

Considering mixed integer linear programming (MILP) models for just-in-time flow shop environments, we can highlight just some works. Ronconi and Birgin (2012) developed and implemented several MILP formulations for the earliness-tardiness problem, including no storage constraints and blocking in-process. M'Hallah (2014) presented new MILP models for the earliness-tardiness flow shop where idle time can be inserted. And Fuchigami et al. (2018) proposed a MILP model and constructive heuristics for flow shop with number of just-in-time jobs maximization.

The only paper addressing a MILP model and common due date for flow shop was of Chandra et al. (2009). Although we can find these MILP formulations for just-in-time flow shop problems, none of them considers the due date assignment, i.e., the due date as a variable of the problem. Therefore the present paper aims to fulfil this research gap.

3 Problem description

In the considered $F_m|prmu, d|ETd$ problem, we consider n independent job to be processed on m machines in series. Each job J_j, $j = 1, \ldots, n$, is available for processing at time zero and requires p_{jk} time units, $j = 1, \ldots, n$, $k = 1, \ldots, m$, for processing on each machine M_k, $k = 1, \ldots, m$, which is known beforehand. It is also assumed that: all jobs are available for processing; all jobs follow the same predefined order of operations; no preemption or interruption is allowed; no job can be processed by more than one machine at the same time, and no machine can process more than one operation at the same time; all jobs must visit all machines with strictly positive processing time on all of the machines (no skipping machines).

Let denote d the common due date for all jobs and $\pi = (J_{[1]}, J_{[2]}, \ldots, J_{[n]})$ the sequence of n jobs. The subscript $[j]$ denotes que job occupying in the j^{th} position in the sequence π. As $F_m|prmu, d|ETd$ is a permutation scheduling, the sequence of the jobs on all machines is the same. For a given sequence π, the completion time of J_j on machine M_k is denoted by C_{jk}. The earliness of each job J_j is defined as $E_j = \max\{d - C_{jm}, 0\}$ and the tardiness of each job J_j as $T_j = \max\{C_{jm} - d, 0\}$, $j = 1, \ldots, n$.

There are three cost components: for earliness, tardiness and delaying the due date. The unit earliness penalty is denoted by w^E, i.e. if a job is early by an amount E, it incurs an earliness cost of $w^E E$. Similarly, the unit tardiness penalty is represented by w^T, i.e. if a job is tardy by an amount T, it incurs a tardiness cost of $w^T T$. The unit due date penalty is denoted by

w^d, i.e. the cost of a due date at time d is $w^d d$. The objective is look for the sequence π and the due date that minimize the cost function: $ETd = w^E \sum_{j=1}^{n} E_j + w^T \sum_{j=1}^{n} T_j + w^d d$.

The optimal solution consists of a sequence π^* and a due date d^* with the optimal cost ETd^*. Following notation is used throughout this paper:

n	number of jobs
m	number of machines
J_j	job j
p_{jk}	processing time of job J_j in the machine M_k (k^{th} operation)
C_h	completion time of job at the position h in the last machine (formulation 1)
C_{hk}	completion time of job at the position h in the machine M_k (formulation 2)
C_{jk}	completion time of job J_j in the machine M_k (formulations 3 and 4)
E_j	earliness of job J_j
T_j	tardiness of job J_j
ET	total earliness and tardiness of all jobs
I_{hk}	idletime of the machine M_k before the job at the position h (formulation 1)
W_{hk}	waiting time of the job at the position h in the machine M_k (formulation 1)
d	common due date of jobs
d^*	optimal due date for all jobs
V_k	a large enough value for the load of the machine M_k (formulations 3 and 4)

4 Formulations for flow shop with due date assignment

According to Ronconi and Birgin (2012), Tseng et al. (2004), and Pan (1997), the classic permutation flow shop environment with makespan minimization is better represented by Wagner's (1959), Manne's (1960) and Wilson's (1989) formulations. Moreover, Liao and You (1992) proposed an improvement to Manne's model. Thereby, we introduced here four new formulations based on these four previous works for the problem treated in the present research. Formulations 1 and 2 own position-based variables and the classic assignment constraints; formulations 3 and 4 hold sequence-based variables.

4.1 Formulation 1 – position-based model with job adjacency machine-linkage constraints

Wagner (1959) introduced originally a position-based model for a three-machine flow shop problem. Baker (1974) and Stafford Jr. (1988) extended it to accommodate problems with more than three machines. We adapted this modelling strategy for the due date assignment problem as follows. Let $x_{jh} \in \{0, 1\}$, j=1, …, n, h=1, …, n, equal to 1 if job J_j is scheduled at the position h of the sequence and 0 otherwise, C_h the completion time of job at the position h in the last machine, I_{hk} the idletime of the machine M_k before the job at the position h and W_{hk} the waiting time of the job at the position h in the machine M_k.

$$Min\ ETd = w^E \sum_{j=1}^{n} E_j + w^T \sum_{j=1}^{n} T_j + w^d d \quad (1)$$

subject to

$$\sum_{h=1}^{n} x_{jh} = 1, \qquad j = 1, \dots, n, \quad (2)$$

$$\sum_{j=1}^{n} x_{jh} = 1, \qquad h = 1, \dots, n, \quad (3)$$

$$C_1 = \sum_{k=1}^{n} \sum_{j=1}^{n} p_{jk} x_{j1} + W_{1k}, \quad (4)$$

$$C_h = C_{h-1} + I_{hm} + \sum_{k=1}^{n} \sum_{j=1}^{n} p_{jm} x_{jh}, \qquad h = 2, \dots, n, \quad (5)$$

$$I_{h+1,k} + \sum_{j=1}^{n} p_{jk} x_{j,h+1} + W_{h+1,k} = W_{hk} + \sum_{j=1}^{n} p_{j,k+1} x_{jh} + I_{h+1,k+1}, \quad h = 1, \dots, n-1, k = 1, \dots, m-1, \quad (6)$$

$$d + T_h - E_h = C_h, \qquad h = 1, \dots, n, \quad (7)$$

$$x_{jh} \in \{0,1\}, j = 1, \dots, n, h = 1, \dots, n,\ C_h \geq 0, E_h \geq 0, T_h \geq 0, h = 1, \dots, n, I_{hk} \geq 0, W_{hk} \geq 0, h = 1, \dots, n, k = 1, \dots, m, d \geq 0. \quad (8)$$

Figure 1 – Mathematical formulation 1

Equation (1) represents the objective function to be minimized as described previously. Constraints (2) and (3) are the classical assignment problem, with (2) ensuring that each job is assigned to just one sequence position, while (3) ensures that each sequence position is filled with only one job. Constraints (4) and (5) are involved in obtaining the completion times of each job, where (4) applies only to the job ordered in the first position, whose completion time depends on its own processing times and the waiting time between the machines. Constraint (5) is general, based on the sum of the completion time of the previous job, the idle time of the last machine between two consecutive jobs and the processing time of the job. Constraint (6) expresses the relationship among several times, such as processing, waiting and idle times, among machines and jobs. Constraint (6) is termed job-adjacency machine-linkage by Tseng et al. (2004). Generally, tardiness and earliness are calculated through the following restrictions, respectively: $T_j \geq d - C_j$ and $E_j \geq C_j - d$, $j = 1, \ldots, n$. Alternatively, we calculate tardiness and earliness by mean of the coupling constraint (7). And constraint (8) is the domain of the variables.

4.2 Formulation 2 – position-based model with inequality constraints

Wilson (1989) also presented a position-based model but instead of the job-adjacency machine-linkage property, he employed the inequality constraints. Let $x_{jh} \in \{0, 1\}$, $j=1, \ldots, n$, $h=1, \ldots, n$, equal to 1 if job J_j is scheduled at the position h of the sequence and 0 otherwise and C_{hk} the completion time of job at the position h in the machine M_k.

$$Min\ ETd = w^E \sum_{j=1}^{n} E_j + w^T \sum_{j=1}^{n} T_j + w^d d \quad (9)$$

subject to

$$\sum_{h=1}^{n} x_{jh} = 1, \qquad j = 1, \ldots, n, \quad (10)$$

$$\sum_{j=1}^{n} x_{jh} = 1, \qquad h = 1, \ldots, n, \quad (11)$$

$$C_{11} \geq \sum_{j=1}^{n} p_{j1} x_{j1}, \quad (12)$$

$$C_{hk} \geq C_{h,k-1} + \sum_{j=1}^{n} p_{jk} x_{jh}, \qquad h = 1, \ldots, n, k = 2, \ldots, m, \quad (13)$$

$$C_{hk} \geq C_{h-1,k} + \sum_{j=1}^{n} p_{jk} x_{jh}, \qquad h = 2, \ldots, n, k = 1, \ldots, m, \quad (14)$$

$$d + T_h - E_h = C_{hm}, \qquad h = 1, \ldots, n, \quad (15)$$

$$x_{jh} \in \{0,1\}, j = 1, \ldots, n, h = 1, \ldots, n,\ C_{hk} \geq 0, h = 1, \ldots, n,$$
$$k = 1, \ldots, m, E_h \geq 0, T_h \geq 0, h = 1, \ldots, n, d \geq 0. \quad (16)$$

Figure 2 – Mathematical formulation 2

Equation (9) represents the objective function. Constraints (10) and (11) are analogous to the (2) and (3) in the Formulation 2, with the assignment problem constraints. Constrains (12) to (14) impose the rules for the completion time of each job on each machine. Constraint (12) simply indicates that the completion time of the first job on the first machine must be at least its processing time. Constraint (13) expresses that between the completion time of a job on two consecutives machines, there must enough time for the job to be processed on the first machine. Constraint (14) indicates that between the completion times of consecutive jobs on a machine, there must be enough time for the first one to be processed. Constraint (15) is the same as (7) in the Formulation 2, with the coupling constraint for calculating tardiness and earliness of each job. And constraint (16) is the domain of the variables.

4.3 Formulation 3 – sequence-based model with disjunctive constraints

Manne (1960) proposed a programming model for the general job shop problem with sequence-based decision variables and disjunctive constraints, i.e., only one of each pair of

constraints can hold, so that job J_i either precedes job J_j somewhere in the sequence, or it does not, thus implying that job J_j precedes job J_i. Let $x_{ij} \in \{0, 1\}$, i=1, …, n, j=1, …, n, equal to 1 if job J_i is scheduled before job J_j anywhere in the sequence and 0 otherwise, C_{jk} the completion time of job J_j in the machine M_k and V_k a large enough value for the load of the machine M_k.

$$Min\ ETd = w^E \sum_{j=1}^{n} E_j + w^T \sum_{j=1}^{n} T_j + w^d d \quad (17)$$

subject to

$$C_{ik} - C_{jk} + V_k x_{ij} \geq p_{ik}, \quad i = 1, \dots, n, j = 1, \dots, n, i \neq j, k = 1, \dots, m, \quad (18)$$

$$C_{jk} - C_{ik} + V_k (1 - x_{ij}) \geq p_{jk}, \quad i = 1, \dots, n, j = 1, \dots, n, i \neq j, k = 1, \dots, m, \quad (19)$$

$$C_{j1} \geq p_{j1}, \quad j = 1, \dots, n, \quad (20)$$

$$C_{j,k+1} \geq C_{jk} + p_{j,k+1}, \quad j = 1, \dots, n, k = 1, \dots, m-1, \quad (21)$$

$$d + T_h - E_h = C_{jm}, \quad h = 1, \dots, n, \quad (22)$$

$$x_{ij} \in \{0,1\}, i = 1, \dots, n, j = 1, \dots, n,\ C_{jk} \geq 0, j = 1, \dots, n, k = 1, \dots, m, E_j \geq 0, T_j \geq 0, j = 1, \dots, n, d \geq 0. \quad (23)$$

Figure 3 – Mathematical formulation 3

Equation (17) represents the objective function. Expressions (18) and (19) are the paired disjunctive constraints which ensure that only one operation is processed on each machine at any given time and that some order must exist between different jobs on the same machine. Constraints (20) and (21) deal with completion times of individual jobs; constraint (21) indicates that the completion time of a job on the first machine must be at least its processing time while constraint (21) expresses that between the completion time of a job on two consecutive machines there must be enough time to process the job on the first one. Constraint (22) is the same as in the previous formulations, with the coupling constraint for calculating tardiness and earliness of each job. And constraint (23) is the domain of the variables.

4.4 Formulation 4 – sequence-based model with surplus variables

Liao and You (1992) algebraically combined each pair of the disjunctive constraints from the Manne's (1960) model into a single equality constraint that they set equal to a surplus variable Q_{ijk}, which accounts for the precedence relationship of jobs J_i and J_j on machine M_k. In order to ensure feasibility, they added a boundary constraint on the surplus variables Q_{ijk} (Ronconi and Birgin, 2012; Tseng et al., 2004). Let $x_{ij} \in \{0, 1\}$, i=1, …, n, j=1, …, n, equal to 1 if job J_i is scheduled before job J_j anywhere in the sequence and 0 otherwise, C_{jk} the completion time of job J_j in the machine M_k and V_k a large enough value for the load of the machine M_k.

$$Min\ ETd = w^E \sum_{j=1}^{n} E_j + w^T \sum_{j=1}^{n} T_j + w^d d \quad (24)$$

subject to

$$Q_{ijk} = C_{ik} - C_{jk} + V_k x_{ij} - p_{ik}, \quad i = 1, \dots, n, j = 1, \dots, n, i \neq j, k = 1, \dots, m, \quad (25)$$

$$Q_{ijk} \leq V_k - p_{jk} - p_{ik}, \quad i = 1, \dots, n, j = 1, \dots, n, i \neq j, k = 1, \dots, m, \quad (26)$$

$$C_{j1} \geq p_{j1}, \quad j = 1, \dots, n, \quad (27)$$

$$C_{j,k+1} \geq C_{jk} + p_{j,k+1}, \quad j = 1, \dots, n, k = 1, \dots, m-1, \quad (28)$$

$$d + T_h - E_h = C_{jm}, \quad h = 1, \dots, n, \quad (29)$$

$$x_{ij} \in \{0,1\}, i = 1, \dots, n, j = 1, \dots, n,\ C_{jk} \geq 0, j = 1, \dots, n, k = 1, \dots, m, E_j \geq 0, T_j \geq 0, j = 1, \dots, n, d \geq 0. \quad (30)$$

Figure 4 – Mathematical formulation 4

Equation (24) represents the objective function. Expression (25) is the surplus variable whose statement can be found in Ronconi and Birgin (2012). Inequality (26) is a lower and upper bound constraint for the surplus variable. Thus, to solve subproblems of the branch and bound tree, the bounded simplex method can be used and achieving an optimal solution may be easier. It is worth noticing that with these new relationships, there is a reduction of one constraint for each pair of jobs in each machine, but it leads to an increase in variables Q_{ijk} in the same amount (Ronconi and Birgin, 2012). Constraints (27) to (30) are the same as in Formulation 3.

5 Computational experiments

In this section, we aim to compare the presented formulations based on the performance of the commercial solver CPLEX when applied to the set of Taillard (1993) instances varying the number of jobs n and the number of machines m according to the Table 1. The test set consist of 120 instances with different sizes n x $m \in$ {20x5, 20x10, 20x20, 50x5, 50x10, 50x20, 100x5, 100x10, 100x20, 200x10, 200x20, 500x20}.

We set the value of the big V_k parameter (usually referred in the literature as big M but here we differentiate it of the "machines M_k") in the formulations 3 and 4 as $V_k = V_{k-1} + \sum_{j=1}^{n} p_{jk}$, with $V_0 = 0$, as considered by Ríos-Mercado and Bard (1998). As the main interest here is the behavior of the variable d, i.e., the common due date assignment, the values of parameters w^E, w^T and w^d where equally set to 1.

Formulations were written and implemented in the syntax of the Julia/JuMP modelling language using the environment JuliaPro 0.6.4.1. It was solved with the branch-and-cut algorithm included in the IBM-CPLEX 12.8.0. All the computational experimentation was conducted on a Pentium Intel Core i7 with 1.99GHz processor, 16.0GB RAM and Windows Operation System. The CPU time where limited to 1 hour of execution (3600 seconds).

Tables 1 to 3 show the detailed and summarized results. Table 1 presents the optimality gap (in percentage) and the CPU time (in seconds) spent for each instance and formulation. The **bold** values mean the optimal solutions of instances and formulations and the symbol "-" indicates no solution found.

Table 1 – Optimality gap (%) and CPU time (in seconds) for each instance and formulation

instance	size (n x m)	Formulation 1		Formulation 2		Formulation 3		Formulation 4	
		Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
ta001	20 x 5	**0.00**	**2.66**	**0.00**	**3.44**	72.51	3602.33	71.04	3602.15
ta002		**0.00**	**2.38**	**0.00**	**1.96**	75.38	3602.81	76.90	3601.81
ta003		**0.00**	**1.47**	**0.00**	**0.92**	70.98	3600.09	73.79	3619.46
ta004		**0.00**	**2.30**	**0.00**	**0.64**	75.74	3600.37	72.02	3600.09
ta005		**0.00**	**3.07**	**0.00**	**1.84**	74.91	3600.11	73.47	3600.07
ta006		**0.00**	**0.96**	**0.00**	**0.58**	73.33	3600.12	73.58	3600.05
ta007		**0.00**	**0.46**	**0.00**	**0.28**	72.69	3600.09	72.41	3600.03
ta008		**0.00**	**0.89**	**0.00**	**1.73**	72.08	3600.14	69.62	3600.05
ta009		**0.00**	**0.77**	**0.00**	**0.61**	70.34	3600.11	67.23	3601.75
ta010		**0.00**	**0.97**	**0.00**	**0.63**	70.37	3602.14	71.85	3600.09
ta011	20 x 10	**0.00**	**9.78**	**0.00**	**5.10**	67.01	3600.14	67.49	3600.07
ta012		**0.00**	**660.50**	**0.00**	**242.03**	72.95	3600.10	67.40	3600.06
ta013		**0.00**	**7.19**	**0.00**	**5.56**	67.02	3600.66	72.47	3600.08
ta014		**0.00**	**7.87**	**0.00**	**2.79**	65.65	3600.12	66.99	3600.10
ta015		**0.00**	**12.52**	**0.00**	**7.62**	71.87	3600.16	68.80	3600.09
ta016		0.00	**9.05**	**0.00**	**4.90**	**64.36**	3600.81	65.48	3600.53
ta017		0.00	**1.84**	**0.00**	**2.86**	**69.14**	3600.11	69.91	3600.23
ta018		0.00	**53.60**	**0.00**	**52.97**	**65.04**	3600.15	68.36	3600.30
ta019		0.00	**2.37**	**0.00**	**4.26**	**69.13**	3600.84	60.47	3600.17
ta020		0.00	**7.54**	**0.00**	**5.05**	**68.56**	3600.09	63.10	3600.59

(to continue)

Table 1 – Optimality gap (%) and CPU time (in seconds) for each instance and formulation *(cont.)*

instance	size (n x m)	Formulation 1		Formulation 2		Formulation 3		Formulation 4	
		Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
ta021	20 x 20	**0.00**	**1137.22**	**0.00**	**1330.07**	58.82	3600.33	64.16	3600.14
ta022		**0.00**	**338.48**	**0.00**	**162.64**	61.48	3600.19	65.08	3600.39
ta023		**0.00**	**1019.42**	**0.00**	**431.86**	64.29	3600.37	63.36	3600.16
ta024		**0.00**	**1204.42**	**0.00**	**1318.29**	64.26	3600.36	63.07	3600.34
ta025		**0.00**	**1242.86**	**0.00**	**2386.84**	62.90	3600.17	66.92	3600.15
ta026		**0.00**	**415.97**	**0.00**	**577.17**	60.69	3600.32	63.67	3600.44
ta027		**0.00**	**3344.01**	**0.00**	**1638.07**	66.27	3600.30	65.66	3600.14
ta028		**0.00**	**212.03**	**0.00**	**147.17**	62.69	3600.12	64.03	3600.36
ta029		**0.00**	**454.43**	**0.00**	**880.36**	64.24	3600.25	93.35	3600.33
ta030		**0.00**	**989.36**	**0.00**	**263.48**	63.98	3600.25	94.55	3600.38
ta031	50 x 5	**0.00**	**4.82**	**0.00**	**3.77**	93.03	3600.35	92.57	3600.53
ta032		**0.00**	**33.73**	**0.00**	**43.87**	94.61	3606.69	94.03	3600.34
ta033		**0.00**	**15.95**	**0.00**	**17.20**	93.51	3600.34	93.39	3600.51
ta034		**0.00**	**123.45**	**0.00**	**103.25**	93.48	3600.33	94.36	3600.35
ta035		**0.00**	**54.15**	**0.00**	**26.25**	94.27	3603.29	93.85	3600.28
ta036		**0.00**	**218.17**	**0.00**	**297.95**	94.47	3600.49	93.54	3600.38
ta037		**0.00**	**31.59**	**0.00**	**82.48**	94.31	3600.67	93.62	3600.53
ta038		**0.00**	**156.77**	**0.00**	**137.87**	94.08	3600.42	93.84	3600.39
ta039		**0.00**	**25.65**	**0.00**	**21.70**	93.45	3600.23	92.63	3600.77
ta040		**0.00**	**8.98**	**0.00**	**7.12**	94.08	3600.20	92.86	3600.52
ta041	50 x 10	0.17	3601.28	0.19	3600.82	93.08	3601.79	92.92	3600.50
ta042		0.27	3600.99	0.26	3601.36	92.88	3600.36	91.67	3600.65
ta043		0.12	3600.45	0.12	3615.00	92.94	3600.33	92.39	3600.85
ta044		0.15	3627.61	0.17	3600.10	92.86	3601.97	91.43	3600.95
ta045		0.30	3601.00	0.30	3603.26	93.33	3600.46	92.35	3600.98
ta046		0.36	3626.54	0.33	3601.95	92.24	3600.76	92.86	3600.61
ta047		0.25	3600.80	0.21	3611.81	92.39	3601.03	92.28	3600.55
ta048		0.44	3602.77	0.41	3601.10	93.29	3601.88	93.02	3600.66
ta049		0.29	3625.96	0.47	3608.75	93.18	3600.92	91.66	3602.03
ta050		0.17	3600.17	0.16	3600.19	93.18	3601.98	92.44	3602.22
ta051	50 x 20	1.50	3609.56	1.22	3609.18	91.35	3607.55	91.73	3603.12
ta052		0.96	3600.71	0.92	3601.28	92.16	3600.80	91.43	3602.48
ta053		1.22	3600.16	1.23	3652.38	91.91	3603.11	90.48	3602.01
ta054		1.00	3602.48	1.18	3627.46	91.60	3614.74	90.05	3601.67
ta055		0.88	3600.72	0.80	3606.31	91.81	3605.70	91.39	3600.84
ta056		1.41	3600.77	1.62	3600.14	89.93	3608.71	91.37	3602.32
ta057		1.35	3600.11	1.44	3631.94	90.68	3606.69	90.46	3601.16
ta058		1.14	3601.91	1.18	3604.90	91.87	3603.58	90.59	3601.69
ta059		1.29	3602.52	1.54	3627.09	90.46	3603.39	97.64	3608.94
ta060		1.51	3600.56	1.55	3600.91	91.16	3600.91	98.07	3604.32
ta061	100 x 5	**0.00**	**2572.64**	**0.00**	**1994.35**	98.22	3600.50	97.55	3600.39
ta062		0.03	3600.54	0.03	3602.72	98.89	3600.42	97.72	3606.99
ta063		0.02	3619.20	0.02	3604.60	98.81	3600.68	97.82	3610.00
ta064		0.04	3617.37	0.04	3604.82	98.43	3600.51	98.04	3609.96
ta065		0.02	3610.77	0.02	3605.66	98.72	3600.55	97.92	3600.92
ta066		0.02	3612.41	0.02	3605.89	98.27	3600.46	98.02	3603.00
ta067		0.02	3616.04	0.02	3601.87	98.84	3600.50	98.35	3600.67
ta068		0.02	3610.93	0.02	3614.77	98.48	3601.61	97.85	3606.47
ta069		0.02	3613.23	0.02	3605.99	98.67	3600.67	97.82	3601.08
ta070		**0.00**	**658.74**	**0.00**	**1039.83**	98.57	3600.46	97.88	3600.65
ta071	100 x 10	0.15	3610.50	0.12	3603.03	98.61	3600.90	97.93	3604.41
ta072		0.22	3600.59	0.21	3642.83	98.46	3600.85	98.08	3602.93
ta073		0.15	3602.22	0.17	3623.70	97.85	3600.89	97.21	3600.68
ta074		0.15	3602.75	0.18	3631.77	98.68	3600.97	97.66	3601.47
ta075		0.21	3603.32	0.16	3616.71	97.14	3600.71	97.85	3600.75
ta076		0.12	3600.98	0.13	3623.66	98.33	3600.84	97.76	3604.17
ta077		0.16	3602.38	0.21	3636.04	97.88	3600.70	97.77	3600.83
ta078		0.13	3602.20	0.17	3612.99	98.26	3604.29	97.95	3602.13
ta079		0.18	3601.11	0.17	3627.20	97.99	3600.84	96.62	3606.74
ta080		0.15	3600.82	0.17	3601.56	98.25	3600.72	96.52	3602.06

(to continue)

Table 1 – Optimality gap (%) and CPU time (in seconds) for each instance and formulation *(cont.)*

instance	size (n x m)	Formulation 1		Formulation 2		Formulation 3		Formulation 4	
		Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
ta081	100 x 20	0.56	3603.44	0.60	3631.90	-	-	97.19	3602.35
ta082		0.62	3600.62	0.61	3600.54	-	-	96.96	3601.44
ta083		0.37	3603.67	0.41	3600.33	-	-	96.96	3601.29
ta084		0.58	3600.85	0.61	3600.48	-	-	97.12	3601.33
ta085		0.64	3600.93	0.65	3600.18	-	-	96.86	3601.60
ta086		0.57	3600.97	0.55	3600.23	-	-	96.93	3602.17
ta087		0.81	3603.48	0.89	3600.56	-	-	97.14	3601.60
ta088		0.62	3600.97	0.60	3600.47	-	-	96.76	3601.60
ta089		0.56	3620.02	0.56	3600.54	-	-	99.04	3602.70
ta090		0.78	3601.26	0.74	3620.96	-	-	99.66	3605.15
ta091	200 x 10	0.06	3600.65	0.07	3605.66	-	-	99.11	3604.36
ta092		0.08	3604.65	0.08	3600.58	-	-	99.64	3601.69
ta093		0.07	3606.28	0.07	3600.46	-	-	99.69	3606.81
ta094		0.07	3602.80	0.07	3600.61	-	-	99.94	3602.18
ta095		0.06	3600.62	0.06	3600.50	-	-	99.11	3603.35
ta096		0.08	3600.99	0.08	3600.67	-	-	99.11	3608.17
ta097		0.07	3608.17	0.08	3600.94	-	-	99.73	3602.56
ta098		0.08	3602.85	0.08	3600.85	-	-	99.06	3602.46
ta099		0.09	3600.56	0.10	3601.90	-	-	99.73	3602.56
ta100		0.08	3603.45	0.08	3600.25	-	-	99.06	3602.46
ta101	200 x 20	0.29	3602.59	0.31	3601.03	-	-	-	-
ta102		0.22	3600.60	0.23	3600.98	-	-	-	-
ta103		0.26	3600.73	0.25	3600.85	-	-	-	-
ta104		0.23	3602.70	0.25	3602.28	-	-	-	-
ta105		0.21	3600.57	0.25	3600.84	-	-	-	-
ta106		0.29	3600.50	0.22	3601.00	-	-	-	-
ta107		0.18	3600.72	0.23	3604.27	-	-	-	-
ta108		0.24	3600.61	0.22	3600.86	-	-	-	-
ta109		0.25	3600.76	0.25	3600.91	-	-	-	-
ta110		0.22	3600.59	0.22	3600.85	-	-	-	-
ta111	500 x 20	0.10	3602.68	99.95	3605.24	-	-	-	-
ta112		31.91	3601.93	31.88	3603.19	-	-	-	-
ta113		0.09	3602.46	99.95	3603.89	-	-	-	-
ta114		0.09	3603.12	99.94	3603.67	-	-	-	-
ta115		0.07	3602.82	99.94	3603.64	-	-	-	-
ta116		0.001	3605.95	99.94	3605.37	-	-	-	-
ta117		0.001	3602.01	99.94	3603.67	-	-	-	-
ta118		0.001	3602.66	99.95	3601.93	-	-	-	-
ta119		0.001	3603.59	99.95	3602.57	-	-	-	-
ta120		0.33	3602.56	99.95	3602.85	-	-	-	-

As discussed in the earlier session, two strategies of modelling were employed: position-based and sequence-based formulations. For each of them two different models were implemented based on previous publications in the specific literature. As can be seen in the Table 1, the position-based models (formulations 1 and 2) overcome the sequence-based ones (formulations 3 and 4). The first two achieved the optimality in 42 instances, in bold in the Table 1, while the last two didn't obtained any optimal solution and besides could not find any integer solution in many instances during the considered computation time. For the instances ta116 to ta119 solved by the formulation 1, it was displayed three decimal places to point out the gap.

Formulations 1 and 2 could obtain at least a factible solution for all the instances in the limited CPU time, including the largest classes with 200 jobs and 20 machines and 500 jobs and 20 machines. Formulation 3 found factible solutions for 80 instances (of 120) up to 100 jobs and 10 machines. And formulation 4 solved 100 instances (of 120) up to 200 jobs and 10 machines.

The overall results for the four models by number of instances and the respective percentage is presented in the following Table 2.

Table 2 – Overall results of each formulation

	Formulation 1		Formulation 2		Formulation 3		Formulation 4	
	# inst	% inst	# inst	% inst	# inst	% inst	# inst	% inst
optimal solution	42	65.00	42	65.00	0	100.00	0	100.00
factible solution	78	35.00	78	35.00	80	33.33	100	16.67
no solution	0	100.00	0	100.00	40	66.67	20	83.33

According to the Table 2, the main results of each pair of the same strategy had similar results in terms of number (and percentage) of optimal solutions, factible solutions and instances with no solutions found. Some differences can be analyzed together with the following Table 3, which presents the average gap in percentage and the average CPU time spent in seconds for each of the four formulations.

Table 3 – Average gaps (%) and CPU times (s)

	Formulation 1	Formulation 2	Formulation 3	Formulation 4
average gap (%)	0.49	7.98	84.85	88.00
average time (s)	2468.27	2455.40	(limit)	(limit)

As can be seen in the Tables 2 and 3, results of formulations 1 and 2 are relatively close. The numbers of optimal, factible and no solutions are the same, according to the Table 1, and the average optimality gap for the formulation 2 is a little higher (7.98%) than the formulation 1 (0.49%). The average CPU time for this both formulations were almost the same. Considering only the 42 instances solved to optimality, the average CPU time spent by the formulation 1 was 358.36 seconds and by the formulation 2 was 315.65 seconds; it may indicate formulation 2 achieves the optimal solution slightly faster.

A special feature is difference in the number of factible solutions found in the formulations 3 and 4, both are sequence-based. This can indicate that the valid inequality proposed by Liao and You (1992) as an improvement of the Manne's (1960) model in the classic flow shop with makespan minimization is also efficient to find factible solutions in the problem with due date assignment considered in this research. However, the average optimality gap in the formulation 4, based on Liao and You (1992) model, is a little higher. Considering only the 80 instances with at least one factible solution in both models, the average optimality's gap of the formulation 3 is 84.85% while of the formulation 4 is 85.39%. As in Table 3, the average gap for the 100 instances solved (not to optimality) by the formulation 4 is 88.00%.

Finally, this experiment demonstrated the applicability of the mathematical models to the problem approach, highlighting those with better computational efficiency.

6 Conclusion

The present paper treated the flow shop with due date assignment problem which can be found in many real applications like maintenance activities, wafer fabrication, logistic systems among others. As few works in the specific literature dealt with this problem and none of them had proposed mathematical models, four new formulations were presented in this work based on published papers.

Four different strategies were employed – two of them with position-based variables (with job-adjacency machine-linkage property and inequality constraints) and the other two with sequence-based variable (with disjunctive constraints and surplus variables). An expressive computation experimentation was conducted with a modern modelling language, the Julia/JuMP (Julia for Mathematical Programming).

Computational results with a well-known instances benchmark of the literature with majority large problems indicate the position-based strategies are more efficient in terms of

optimality gap and CPU time consuming than the sequence-based strategies. Also pointed out the efficacy of the valid inequality evaluated.

For future works, it could be explored other values of the weights in the objective function and considered it as multi-objective with a Pareto frontier treatment, for example. It could be implemented different formulation strategies for the problem, as time-indexed, and employed other optimization solvers. Also the behavior of the models could be analyzed with further data instances.

Acknowledgements The author thanks the supporting by CAPES (Coordination for the Improvement of Higher Education Personnel, Brazil) and FAPEG (Goias State Research Foundation, Brazil), and the careful reading of the reviewers and their relevant contributions.

References

Alvarez-Valdes, R.; Crespo, E.; Tamarit, J. M.; Villa, F. Minimizing weighted earliness-tardiness on a single machine with a common due date using quadratic models. Top, v. 20, p. 754-767, 2012.

Baker, K. R.; Scudder, G. D. Sequencing with earliness and tardiness penalties: a review. Operations Research, v. 38, p. 22-36, 1990.

Baker, K.R. Introduction to sequencing and scheduling. New York: Wiley, 1974.

Birman, M.; Mosheiov, G. A note on a due-date assignment on a two-machine flow-shop. Computers & Operations Research, v. 31, p. 473-480, 2004.

Chandra, P.; Mehta, P.; Tirupati, D. Permutation flow shop scheduling with earliness and tardiness penalties. International Journal of Production Research, v. 47, p. 5591-5610, 2009.

Chen, T. Internal due date assignment in a wafer fabrication factory by effective fuzzy-neural approach. Journal of Applied Mathematics, v. 2013, p. 1-13, 2013.

Cheng, T.C.E.; Gupta, M.C. Survey of scheduling research involving due date determination decisions. European Journal of Operational Research, v. 38, p. 156-166, 1989.

Conway, R. W. Priority dispatching and job lateness in a job shop. The Journal of Industrial Engineering, v. 16, p. 228-237, 1965.

De, P.; Ghosh, J. B.; Wells, C. E. Optimal due-date assignment and sequencing. European Journal of Operational Research, v. 57, p. 323-331, 1992.

Fuchigami, H. Y.; Sarker, R.; Rangel, S. Near-optimal heuristics for just-in-time jobs maximization in flow shop scheduling. Algorithms, v. 11, p. 1-17, 2018.

Gordon, V. S.; Proth, J.- M.; Strusevich, V. A. Scheduling with due date assignment. In Leung, J. Y.- T. Handbook of Scheduling: algorithms, models, and performance analysis. Boca Raton, Champan & Hall/CRC, 2004, Chapter 21.

Gordon, V.; Proth, J.-M.; Chu, C. A survey of the state-of-the-art common due-date assignment and scheduling research. European Journal of Operational Research, v. 139, p. 1-25, 2002.

Gordon, V.; Proth, J.-M.; Chu, C. Due date assignment and scheduling: SLK, TWK and other due date assignment models. Production Planning & Control, v. 13, p. 117-132, 2002.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; Rinooy Kan, A. H. G. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, v. 5, p. 287-326, 1979.

Janiak, A.; Janiak, W. A.; Krysiak, T.; Kwiatkowski, T. A survey on scheduling problems with due windows. European Journal of Operational Research, v. 242, p. 347-357, 2015.

Jing, S. Logistics scheduling with due date assignment and batch delivery. International Conference on Management Science & Engineering, p. 502-506, 2013.

Józefowska, J. Just-in-time scheduling: models and algorithms for computer and manufacturing systems. New York: Springer Science, 2007.

Kaminsky, P.; Hochbaum, D. Due date quotation models and algorithms. In Leung, J. Y.- T. Handbook of Scheduling: algorithms, models, and performance analysis. Boca Raton, Champan & Hall/CRC, 2004, Chapter 20.

Kaminsky, P.; Lee, Z. H. Effective on-line algorithms for reliable due date quotation and-large-scale scheduling. Journal of Scheduling, v. 11, p. 187- 204, 2008.

Kaminsky, P.; Lee, Z. H. On-line algorithms for flow shop due date quotation. University of California, Berkeley (California, USA), 2002. http://www.ieor.berkeley.edu/~kaminsky/papers/ddq_flowshop.pdf.

Keskinocak, P.; Tayur, S. (2004). Due Date Management Policies. In Handbook of Quantitative Supply Chain Analysis: Modeling in the eBusiness Era. p. 485 - 547. Kluwer Academic Publishers.

Koulamas, C. The total tardiness problem: review and extensions. Operations Research, v. 42, p. 1025-1041, 1994.

Lauff, V.; Werner, F. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: a survey. Mathematical and Computer Modelling, v. 40, p. 637-655, 2004.

Liao, C.J.; You, C.T. An improved formulation for the job-shop scheduling problem. Journal of the Operational Research Society, v. 43, p. 1047-1054, 1992.

M'Hallah, R. Minimizing total earliness and tardiness on a permutation flow shop using VNS and MIP. Computers & Industrial Engineering, v. 75, p. 142-156, 2014.

Manne, A.S. On the job-shop scheduling problem. Operations Research, v. 8, p. 219-223, 1960.

Mohamad, N.H.; Said, F. Solving single machine scheduling problem with common due date. Business Management Dynamics, v. 1, n. 4, p. 63-72, 2011.

Mor, B. Minmax scheduling problems with common due-date and completion time penalty. Journal of Combinatorial Optimization, p. 1-22, 2018.

Mosheiov, G.; Oro, D. Due-date assignment and maintenance activity scheduling problem. Mathematical and Computer Modelling, v. 44, p. 1053-1057, 2006.

Pan, C.H. A study of integer programming formulations for scheduling problems. International Journal of Systems Science, v. 28, p. 33-41, 1997.

Perez-Gonzalez, P.; Framinan Torres, J. M.; Gonzalez-R., P. L.; Blanco, J. M. L.; Ruiz-Usano, R. Flowshop scheduling problems with due date related objectives: a review of the literature. International Conference on Industrial Engineering and Industrial Management, p. 1488-1497, 2009.

Perez-Gonzalez, P.; Framinan, J. M. Assessing scheduling policies in a permutation flowshop with common due dates. International Journal of Production Research, v. 53, p. 5742-5754, 2015.

Perez-Gonzalez, P.; Framinan, J. M. Setting a common due date in a constrained flow shop: a variable neighbourhood search approach. Computers & Operations Research, v. 37, p. 1740-1748, 2010.

Ragatz, G. L.; Mabert, V. A. A framework for the study of due date management in job shops. International Journal of Production Research, v. 22, p. 685-695, 1984.

Rajasekera, J. R.; Murr, M. R.; So, K. C. A due-date assignment model for a flow shop with application in a lightguide cable shop. Journal of Manufacturing Systems, v. 10, p. 1-7, 1991.

Ríos-Mercado, R.Z.; Bard, J.F. Computational experience with a branch-and-cut algorithm for flow shop scheduling with setups. Computers & Operations Research, v. 25, p. 351-366, 1998.

Ríos-Solís, Y.A.; Ríos-Mercado, R.Z. Just-in-Time Systems. New York: Springer Sciences, 2012.

Ronconi, D.P.; Birgin, E.G. Mixed-integer programming models for flow shop scheduling problems minimizing the total earliness and tardiness. In Ríos-Solís, Y.A. e Ríos- Mercado, R.Z. (Eds.) Just-in-Time Systems, Springer Sciences, New York, 2012.

Samaghandi, H. A particle swarm optimization for the no-wait flow shop problem with due date constraints. International Journal of Production Research, v. 53, p. 2853-2870, 2015.]

Samarghandi, H.; Behroozi, M. On the exact solution of the no-wait flor shop problem with due date constraints. Computers and Operations Research, v. 81, p. 141-159, 2017.

Sen, T.; Gupta, S. K. A state-of-art survey of static scheduling research involving due dates. Omega – The International Journal of Management Science, v. 12, p. 63-76, 1984.

Shabtay, D.; Steiner, G. Scheduling to maximize the number of just-in-time jobs: a survey. In Ríos-Solís, Y.A. and Ríos-Mercado, R.Z. (Eds.) Just-in-Time Systems, Springer Sciences, New York, 2012.

Stafford Jr., E.F. On the development of a mixed-integer linear programming model for the flowshop sequencing problem. Journal of the Operational Research Society, v. 39, p. 1163-1174, 1988.

Taillard, E. Benchmark for basis scheduling problems. European Journal of Operational Research, v. 64, p. 278-285, 1993.

Tseng, F.T.; Stafford Jr., E.F.; Gupta, J.N.D. An empirical analysis of integer programming formulations for the permutation flow shop. Omega, v. 32 p. 285-293, 2004.

Vallada, E.; Ruiz, R.; Minella, G. Minimising total time in the *m*-machine flowshop problem: a review and evaluation of heuristics and metaheuristics. Computers & Operations Research, v. 35, p. 1350-1373.

Wagner, H.M. An integer linear-programming model for machine scheduling. Naval Research Logistic, v. 6, p. 131-140, 1959.

Wilson, J.M. Alternative formulations of a flow-shop scheduling problem. Journal of the Operational Research Society, v. 40, p. 395-399, 1989.

Xiong, X.; Wang, D.; Cheng, T. C. E.; Wu, C. -C.; Yin, Y. Single-machine scheduling and common due date assignment with potential machine disruption. International Journal of Production Research, v. 56, p. 1345-1360, 2018.

MISTA 2019

Feature-based tuning of a Tactical Berth Allocation heuristic

Emmanouil Thanos · Thomas Van den Bossche · Haroldo Gambini Santos · Wim Vancroonenburg · Greet Vanden Berghe

Abstract Efficient berth allocation is capable of reducing container transshipments and discharge distances in port terminals. When quay crane assignment decisions are also involved, their effect may vary for different settings and layout characteristics. The present paper combines the Tactical Berth Allocation problem with Specific Quay Crane Assignments and proposes a heuristic approach for minimizing container transfer distances. In the absence of existing approaches for the tactical version of the integrated problem when specific crane assignments are considered, the analysis focuses on the tuning effort of the developed algorithm. A variety of methods are implemented for the incorporation of crane assignments and an approximation scheme is also proposed to mitigate the impact of the quadratic objective function. An integrated set of instances which covers a wide range of diverse layout specifications is generated and made publicly available. Together with the benchmark results, an extensive methodology is applied to discover efficient configurations of the algorithm with respect to different instance characteristics. Experimental results demonstrate the significance of instance-based tuning for the underlying problem.

1 Introduction

Container handling operations constitute a critical component of maritime freight transportation. Efficient berth allocation decisions are capable of reducing container transshipments and discharge distances, facilitating traffic procedures and providing terminals with a competitive advantage. Depending on vessel traffic congestion and container processing demands, quay crane resources limitations may impose great difficulties in providing high-quality berthing schedules. Although transshipment-related objectives typically consider longer planning horizons, terminal managers must fre-

Emmanouil Thanos, Thomas Van den Bossche, Wim Vancroonenburg, Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS

Haroldo Gambini Santos
Federal University of Ouro Preto, Department of Computing

E-mail: emmanouil.thanos@kuleuven.be

quently respond to unexpected schedule modifications which require prompt decisions when negotiating with shipping lines.

Due to the rapid development of management techniques for port logistics and seaside operations in recent years, the integration of the Berth Allocation (BAP) and Quay Crane Assignment (QCAP) problems has received considerable attention in the academic literature (Bierwirth and Meisel, 2015). In the combined Berth Allocation and Quay Crane Assignment Problem (BACAP) both a berthing position and a number of cranes is assigned to each vessel. When the assignment of specific cranes to each vessel is also considered to provide more realistic schedules then the resulting problem is referred to as the BACASP. Feasible solutions are typically evaluated with respect to vessel handling times, as well as the deviation of their assigned positions from desired ones. For a general review of classifications and integrations of BAP and QCAP, as well as independent developments within the two problems, interested readers are referred to (Bierwirth and Meisel, 2010, 2015).

The Tactical variant of the integrated problem (TBAP) takes into account the terminal's container yard. It is assumed that vessels arrive in a cyclic fashion and the objective is to minimize housekeeping costs generated by transshipment flows, typically in a weekly time horizon. Giallombardo et al. (2010), Vacca et al. (2013) and Lalla-Ruiz et al. (2014) all address the TBAP for discrete quay layouts. Each of these studies employed the concept of QC profiles to encode the number of quay cranes per working shift along with real-world limitations. More recently, Liu et al. (2016) considered transshipment-related quadratic yard costs for the continuous BACAP. Their method, however, also utilizes QC profiles and not individual crane assignments. It thus remains theoretical, not ensuring that a solution is physically realizable.

When berth allocation and time-variant specific crane assignments are addressed simultaneously, the underlying settings typically consider discrete quay layouts or a limited number of candidate berthing positions. Imai et al. (2008) considered the discrete BAP with time-variant assignment of specific QCs through detailed QC movement constraints and proposed a Genetic Algorithm for its solution. Türkoğullari et al. (2016) solved the BACASP in an exact manner for instances containing up to 60 vessels, considering a continuous but discretized quay setting with 24 berth slots and 12 cranes.

Although various discretization policies for quay length sections and time horizons are employed throughout the literature, the effect of granularity on solution quality has rarely been explored. Iris et al. (2015) proposed generalized set partitioning formulations for both time-variant and time-invariant versions of BACAP. They compared their performance for small and medium-sized instances and analyzed the effects of different discretization schemes.

In addition, studies on identical or similar BACAP variants rarely respect a unified convention for length units, time horizons or crane assignment policies. This is mostly due to the lack of a generally accepted benchmark set. One which is compatible with the majority of BAP variants and which fulfils the principles of comparability, unbiasedness, and reproducibility. The need for such a set when it comes to an objective comparison of available solution methods has already been highlighted by Bierwirth and Meisel (2015).

Regarding the effect of QCAP incorporation strategies, both Vacca et al. (2013) and Meisel and Bierwirth (2013) perform comparative analyses to assess the added value of deep integration for their examined settings. Abou Kasm et al. (2019) also investigate the effect of different crane assignment policies on the total schedule derived from their

case study concerning an integrated BAP, QCAP and Quay Crane Scheduling Problem approach for a container terminal located in Abu Dhabi.

However, the variety of quay layouts, terminal layouts, container load sizes and vessel traffic congestion levels which may arise for different ports and settings stresses the fact that such analyses should be instance-based. Given that related approaches are typically intended for real-time decision support systems, it is of great significance to estimate to what degree QCAP integration may contribute toward solution quality improvement for identifiable setting features. By doing so, unnecessary overheads could be avoided via feature-based tuning and method selection strategies. Furthermore, recent research has indicated that there are gains to be realized by combining such analyses with simulated annealing parameter configuration (Bellio et al., 2016; Santos et al., 2019). Although there has been a recent development in algorithm selection approaches for diverse optimization problems (Chand and Wagner, 2016; Messelis and De Causmaecker, 2014), their applicability to BAP variants is yet unknown.

The present paper investigates the integration of continuous Berth Allocation and time-variant Specific Quay Crane Assignment problems with the aim of minimizing container transfer distances throughout the terminal yard. The general setting and formulation of the underlying Tactical Berth Allocation and Specific Quay Crane Assignment Problem (TBACASP) are provided in Section 2.

The algorithm proposed for solving the integrated problem is detailed throughout Section 3. It employs a Remove & Reinsert (R&R) heuristic utilizing spatio-temporal criteria, which have demonstrated significant potential for improving berthing schedules (Li et al., 2015). Four diverse constructive methods are implemented for the reinsertion step, each of which applies a different QCAP incorporation strategy. The heuristic is incorporated into a simulated annealing framework. A delta evaluation approximation scheme is also proposed to moderate the impact of the quadratic transshipment-related objective function upon computation time.

Experimental tests are conducted, the results of which are presented in Section 4. An integrated set of benchmark instances is generated and made publicly available, which covers a wide range of specifications for quay layouts, vessel arrivals, QC features and container loads. The proposed approach is evaluated over the set of instances from an algorithm selection point of view. The iRace package (López-Ibáñez et al., 2016) is employed to investigate the effectiveness of different QCAP incorporation strategies and thus the added value of the integrated and time-variant approach with respect to different setting and instance features. Similar to incorporation strategies, the proposed approximation scheme and the configuration of various parameters, including the considered granularity for berth length, are also analyzed. The most significant conclusions are then documented in Section 5, which also outlines directions for future research.

2 Tactical Berth Allocation Problem with Specific Crane Assignments

The Tactical Berth Allocation Problem with Specific Crane Assignments (TBACASP) considers a port terminal quay along with its container yard. The quay is characterized by its total length L and is possibly separated into distinct parts, in which case it is assumed that each part corresponds to a different continuous berthing zone. The quay illustrated in Figure 1 consists of two berthing zones of $L/2$ length each. An ordered set of bollards B are located across the quay and used by the vessels to moor.

Each bollard $b \in B$ defines a distinct berthing position where a vessel is allowed to moor, meaning that its leftmost part will be coinciding with b. Finally, two consecutive bollards located in the same zone are separated by a fixed distance $u = L/|B|$, which defines the minimum considered length unit.

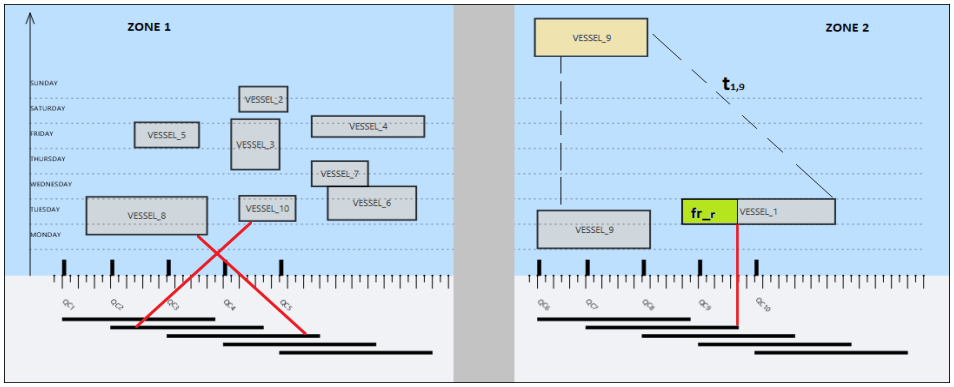


Fig. 1 Terminal and setting configuration. Vessels are represented by rectangles. The vertical axis shows the time during which vessels are present at the quay.

A set C of identical QCs are installed and located uniformly along the quay. Each crane $c \in C$ can operate within a limited range of bollards $B_c = [b_c^{from}..b_c^{to}]$ and is further characterized by a container handling throughput p_c which indicates the maximum number of containers it can process per hour. Range length $|B_c| \cdot u$ and throughput p_c are equal for all available cranes. Finally, the set of cranes whose associated ranges overlap with a crane c is denoted as C_c^N.

Vessels carrying containers arrive at the terminal and they must be assigned to a berthing position in the quay, where they will remain moored for their entire stay. During a vessel's stay a part of its load must be discharged at its mooring position in order for it to be transshipped to other vessels. Likewise, containers which are scheduled to depart with the vessel must also be transferred and loaded to it.

An arrival schedule $< V, T >$ thus consists of a set V of arriving vessels and a set T of container transshipments. Each vessel $v \in V$ is characterized by a tuple

$< start_v, end_v, \ell_v, maxNumCranes_v >$ where :

- $start_v$ is the arrival time of v in hours with respect to the start of time horizon,
- end_v is the departure time of v in hours, similar to $start_v$,
- ℓ_v is the length of v in meters and
- $maxNumCranes_v$ is the maximum number of cranes which may be serving v during its stay.

Each transshipment $t_{v_t^1, v_t^2} \in T$ consists of a tuple $< v_t^1, v_t^2, n_t >$, where $v_t^1 \neq v_t^2 \in V$ and where n_t is the number of containers to be moved from v_t^1 to v_t^2. For each vessel $v \in V$ the set of associated transshipments t with $v_t^1 = v$ or $v_t^2 = v$ is denoted as T_v. The total number of containers which must be processed for v is $N_v = \sum_{t \in T_v} n_t$.

For the purposes of this work it is assumed that the schedule of arrivals is weekly and cyclic, meaning that it is repeated every week. In this manner and without compromising on generality, container transshipment flows concern two consecutive weeks for each pair of vessels and are independent of the arrivals' sequence. An illustrative example is shown in Figure 1 for the two vessels moored at Zone 2.

A Bollard Assignment (BA) a consists of a tuple $< b_a, v_a >$ which indicates the allocation of the berthing position defined by $b_a \in B$ to $v_a \in V$ for the vessel's stay. Each BA a is associated with the ordered set of bollards $B_a = [b_a..b'_a]$ which are covered by v_a's length, with $|B_a| \cdot u \geq \ell_{v_a} > (|B_a| - 1) \cdot u$. It is also characterized by a bollard b_a^{center}, which is the leftmost bollard located closest to the center of B_a. The distance D_{a_1,a_2} between two BAs a_1, a_2 is equal to the distance between $b_{a_1}^{center}$ and $b_{a_2}^{center}$. This distance is considered as known and is equal to the physical distance between the bollards, with an additional penalty d^p if they belong to different berth zones.

The set of all possible BAs is denoted as A. The subset of these BAs which are compatible with vessel $v \in V$ is then denoted as A_v. Two BAs $a_1, a_2 \in A$ are incompatible with each other when i) v_{a_1}'s stay overlaps with v_{a_2}'s [1] and ii) either B_{a_1} overlaps with B_{a_2} or there is no safety distance d^s separating them. Finally, each BA a is associated with a set C_a of compatible cranes, with $c \in C_a \iff B_c \cap B_a \neq \emptyset$.

A crane assignment r indicates the operation of a crane with respect to a vessel and consists of a tuple $< a_r, c_r, start_r, end_r >$, where:

- $a_r \in A$,
- $c_r \in C_{a_r}$,
- $start_r \geq start_{v_{a_r}}$ is the start time of the operation and
- $end_r \leq end_{v_{a_r}}$ is the end time of the operation.

The length fraction of v_{a_r} which is reachable by c_r is denoted as fr_r, with $fr_r = \frac{|B_{c_r} \cap B_{a_r}|}{|B_{a_r}|}$, as illustrated in the example at Zone 2 of the quay in Figure 1. A vessel's stowage plan is assumed to be symmetrical and therefore the number of v_{a_r}'s containers which are processed by r is bounded by $fr_r \cdot N_{v_{a_r}}$ and is equal to
$n_r = \min(fr_r \cdot N_{v_{a_r}}, (end_r - start_r) \cdot p_c)$.
A set R of crane assignments covers a BA a when $\exists R_a \subseteq R$ such that the following three conditions are met:

i) $a_r = a \quad \forall r \in R_a$,
ii) $\sum_{r \in R_a} n_r \geq N_{v_{a_r}}$ and
iii) $\exists r \in R : b \in B_{c_r} \quad \forall b \in B_a$, in other words every part of the vessel is reachable by at least one of the cranes assigned to it.

Two crane assignments r_1, r_2 are crossing when $b_{a_{r_1}}^{center} > b_{a_{r_2}}^{center}$ and $b_{c_{r_1}}^{from} < b_{c_{r_2}}^{from}$, as shown in the example at Zone 1 of the quay in Figure 1. Two crane assignments r_1, r_2 are incompatible with each other when i) they overlap in time and ii) they are either crossing or $c_{r_1} = c_{r_2}$. A set R of crane assignments is valid when it contains no pair of incompatible crane assignments.

A set A^V of valid BAs is complete for a set of vessels V when

i) It contains no pair of incompatible BAs,

[1] In order to accurately model time overlaps in a cyclic schedule, assignments occurring at the start and at the end of the schedule must be also checked against each other. These auxiliary checks are included in the model implementation, however in order to prevent unnecessary overcomplication they are not considered in the proposed formulation.

ii) $\forall v \in V$ there exists exactly one $a \in A^V$ such that $v_a = v$ and
iii) there exists a valid set R_{A^V} of crane assignments which covers all BAs in A^V.

Under a given A^V, the distance D_{v_1,v_2} between two vessels $v_1, v_2 \in V$ is equal to the distance between their assigned BAs a_{v_1}, a_{v_2} in A^V: $D_{v_1,v_2} = D_{a_{v_1},a_{v_2}}$

Given a specific quay layout with a quay of length L, its set of bollards B, a set of cranes C and an arrival schedule $< V, T >$, the TBACASP consists of finding a set A^V of valid BAs which is complete for V and which minimizes the total container transfer distance $TCTD$, calculated using Equation 1 as follows:

$$TCTD = \sum_{t \in T} n_t \cdot D_{v_t^1, v_t^2} \tag{1}$$

3 A heuristic approach

The complexity and size of TBACASP in real-world settings impose great difficulties with respect to finding globally optimal solutions within short computation times. This section therefore introduces a heuristic approach which provides real-time decision support.

The Simulated Annealing (SA) metaheuristic is employed to explore the search space. SA was proposed by Kirkpatrick et al. (1983) and is a probabilistic metaheuristic which is based on an analogy with the cooling procedure of heated atoms in a thermodynamic system.

For a given setting and an arrival schedule $< V, T >$ the proposed approach applies a preprocessing step which involves the generation of all valid BAs A_v for every vessel $v \in V$. Please note that the set B of available bollards depends upon the minimum considered length unit u, which is a parameter of the algorithm. For each BA a the set C_a of compatible cranes is subsequently determined. This step employs an additional parameter $rcc \in (0, 1]$ which defines the minimum required crane coverage, discarding from C_a all cranes c which cover a fraction $\frac{|B_c \cap B_a|}{|B_a|}$ smaller than rcc.

3.1 Solution representation

The search space of complete BA sets is explored and evaluated within the metaheuristic framework working on a direct solution representation. Assuming vessels V are sorted by ascending order of arrival time, a solution S is defined by:

- a vector A^S of length $|V|$, which at each position A^S_v contains the BA $a_v \in A_v$ assigned to v. When v is unassigned, A^S_v is empty.
- for each $v \in V$, a vector R^S_v which at each position $R^S_{v_i}$ contains a crane assignment r associated with v. When v is unassigned, R^S_v is empty.
- a vector V^S_U which contains any unassigned vessels.

3.2 R&R strategy

At each step of the search the algorithm applies an R&R strategy to modify the current solution. In the "Remove" step a set A_{rem} of BAs are removed from current A^S, the

corresponding vessels are added to V_U^S and their associated crane assignment vectors R_v^S are emptied. In the"Reinsert" step one of the following constructive methods is applied to the set of unassigned vessels.

- First-Fit (**FF**): For each unassigned vessel v, A_v is iterated over. If a BA $a \in A_v$ is found which incurs no conflicts with the current solution's already-assigned BAs, the algorithm attempts to assign cranes to it. If a set of crane assignments which covers a is found, BA a is assigned to v and the iteration over A_v ends.
- Best-Fit (**BF**): A_v is iterated over as in FF, but now exhaustively. For each $a \in A_v$ with a covering set of crane assignments, the cost contribution $TCTD(T_v)$ of v under a is computed and the BA with the lowest cost contribution is assigned to v.
- check-Only Best-Fit (**OBF**): OBF follows the same procedure as BF, but instead searches for a covering set of crane assignments only after the cost contribution calculation and only if assignment a is the current best. This constructive method is most suited to congested settings with reduced container loads where finding good-quality BAs poses a greater challenge than finding covering sets of crane assignments.
- Re-do Crane Assignments (**RCA**): This constructive method works the same as OBF, but for each current best BA the crane assignment procedure is conducted for all assigned BAs of the current solution, starting with the leftmost one. This time-consuming step attempts to reduce the possibility of falsely rejecting a complete A^V due to inefficient crane assignment decisions.

Algorithms 1-4 in Appendix A provide the pseudocodes to complement the textual descriptions.

The assignment of cranes to a BA a of the current solution S in the aforementioned methods is achieved by the procedure $AssignCranes(a)$, outlined by way of Algorithm 5 in Appendix A. This fast constructive procedure iterates over all $c \in C_a$, searching for the maximum vacant time interval I of c which is capable of generating a crane assignment $r =< a, c, I^-, I^+ >$ that is compatible with all existing crane assignments. For r to be considered as a candidate crane assignment, the respective interval must overlap with a minimum fraction of v's stay which is defined by the parameter $mcaf \in (0, 1]$.

3.3 Neighbourhoods

Given the constructive method utilized at the"Reinsert" step, which is determined by the parameter $CM \in (FF, BF, OBF, RCA)$ of the algorithm, every R&R iteration modifies the current solution by applying one of the two following neighbourhood operators:

- RANDOM_R&R(n): remove n random BAs from the current solution and reinsert the removed vessels V_{rem} in a random order along with any unassigned ones using $CM(V_{rem} \cup V^U)$. Integer n is randomly selected in $[1, maxNumRem]$, where $maxNumRem$ is a parameter defining the maximum number of removed arrivals.
- SPATIO-TEMPORAL_R&R(n): randomly select two vessels v_1, v_2 which are assigned in the current solution and which overlap in time, such that $\ell_{v_1} \geq \ell_{v_2}$. Find all vessels $V_{overlap}$ which would overlap with v_1 if it were assigned at v_2's starting

position, as shown in Figure 2. Then remove vessels $V_{rem} = (v_1, v_2) \cup V_{overlap}$ and reinsert them along with any unassigned ones in a random order using $CM(V_{rem} \cup V^U)$.

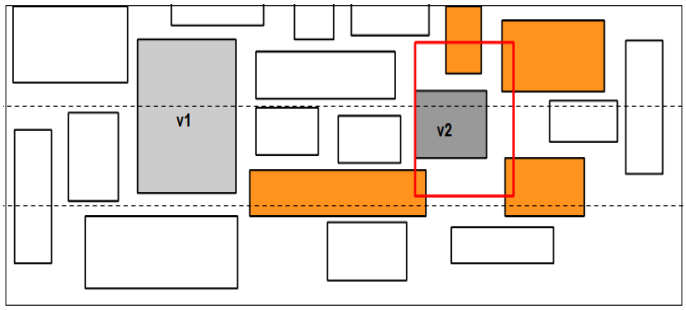


Fig. 2 Spatio-temporal neighbourhood operator

For initial solution S_0, the First-Fit constructive method is applied to the entire set of vessels V. At every step of the search process one of the neighbourhood operators RANDOM_R&R and SPATIO-TEMPORAL_R&R is randomly selected with $(1 - stp)$ and stp probabilities, respectively, where $stp \in (0, 1)$ is a parameter of the algorithm. A neighbour is subsequently generated by the constructive method and its TCTD is calculated. For every unassigned vessel v included in a solution S, a cost penalty $P^U = \ell_v \cdot (end_v - start_v) \cdot p_U$ is added to S's TCTD that is proportional to v's length and duration. The value of multiplier p_U is case dependent and relies on $|V|$ and L. Depending on the resulting cost and the current temperature, SA determines whether a neighbour is accepted.

The SA algorithm which is employed applies a linear cooling schedule. Cooling rate cs is expressed as a fraction of the initial temperature T_0, meaning that after each iteration the temperature is reduced by $dT = (cs \cdot T_0)$. If the temperature drops below 10^{-3} at any point then it is reset to T_0. The pseudocode of SA is also provided in Algorithm 6 in Appendix A for completeness.

Various data-structures which are updated after each iteration are utilized to store, connect and quickly retrieve the vectors associated with the solution. Spatial and temporal overlaps are addressed by special data-structures which provide dedicated functions to handle geometry and time intervals. These functions are based on interval trees and enable a significant speed-up of BA conflict checks, the crane assignment procedure and the spatio-temporal neighbourhood operator.

3.4 Delta approximation scheme

The calculation of the quadratic objective function may significantly lengthen the runtime for the tactical variants of Berth Allocation problems. To moderate its effect an approximation scheme is proposed which considers only those BAs which are affected by the current local move. Delta evaluation is applied to the partial objective representing the transshipments of the corresponding vessels and their sum is utilized to approximate their impact upon the objective function. The exact TCTD computation

Feature	Small	Medium	Large
Quay length L (m)	1000	2000	4000
Number of berthing zones	1,2	1,2	1,2
Crane coverage range $\lvert B_c \rvert \cdot u$ (m)	200,400	600,800	600,800
Number of cranes $\lvert C \rvert$	10,20	20,30	40,60
Number of vessels $\lvert V \rvert$	10,20,25	20,25,30	40,50,60
Provisional Container load N_v^P	800,1000	800,1000	800,1000

Table 1 Instance templates. For each quay length, all possible combinations of the corresponding column were applied, thus generating 48 instances for each of the three groups.

occurs every 100 runs in order to correct accumulated errors. The proposed approximation scheme is most suited for the FF constructive method, where the calculation of the TCTD is proportionally more time-consuming within the search process.

4 Experimental Evaluation

This section presents an extensive experimental evaluation which aims to determine efficient configurations of the proposed approach in both a general as well as an instance-specific context. For this purpose, an integrated set of instances must first be generated in order to obtain a wide range of diverse and realistic conditions.

4.1 Generation of benchmark instances and experimental setup

An instance generator has been developed which enables the generation of problem instances in a controlled manner. The quay layouts were based on the real features of an existing terminal located in Antwerp. Historical data provided by a terminal operator was utilized for the generation of arrival schedules. Various real-world characteristics which were observed in practice, as well as those described in other BACAP benchmarks found in the literature (Park and Kim, 2005; Meisel and Bierwirth, 2009) were also considered.

Three quay lengths were employed: 1000m, 2000m and 4000m. These three quay lengths correspond to small, medium and large-sized instances, respectively. For each of these sets both 1-zone and 2-zone settings were applied. All instance features along with their associated values for all 144 instances are reported in Table 1.

Regarding the positioning of the QCs, please note that both the left edge of the leftmost crane and right edge of the rightmost crane do not coincide with the boundaries of the zone. Therefore, some parts of the quay are unreachable by any crane, as shown in Figure 1. The length of these parts always sums to 40m for both 1-zone and 2-zone instances. For the generation of the distance matrices, bollards were assumed to be positioned linearly as shown in Figure 1. For 2-zone settings, the distance which separates the bollards closest to one another from the two zones was assumed to be 515m.

Available historical data included information concerning $start_v$, end_v, ℓ_v and $maxNumCranes_v$ for a pool of 40 vessels, as well as the transshipments between those vessels for six consecutive weeks. Vessel lengths varied from 70 to 400m and their stay durations from 15 to 72 hours.

For each of the nine possible combinations concerning quay length and number of vessels, vessels were randomly selected from the pool until $|V|$ was reached. For each vessel v, the values of ℓ_v and $maxNumCranes_v$ were not modified. Arrival time $start_v$ was selected at random from within the weekly time horizon and the corresponding departure time end_v was chosen such that the vessel's stay duration remained unchanged. Vessels with $\ell_v > 200$ were impractically large for the smallest quay length and therefore were excluded for small-sized instances. For each quay length, the size $|V|$ of the largest vessel set was selected such that very tightly constrained instances for the BAP were generated. Several runs of the instance generator were often required before arrival times were generated which allowed for feasible, non-overlapping berth allocations for the maximum selected length units.

Once set V was formed, two sets of transshipments were constructed, one for each provisional container load N_v^P indicated in Table 1. For each pair (v_1, v_2) of vessels with $v_1 \neq v_2 \in V$, a transshipment $< v_1, v_2, n_t >$ was generated. The number of containers n_t was randomly selected from the range
$[(1/2 \cdot durCoef \cdot avgCont), (3/2 \cdot durCoef \cdot avgCont)]$, where

$durCoef = (duration(v_1)/averageDuration(V)$ and
$avgCont = N_v^P/(2 * numArrivals - 2)$.

After the construction of set T, a check was performed for each vessel to determine whether there exists a set of crane assignments with $maxNumCranes_v$ cranes which is capable of serving the vessel within its stay. When this check failed for a vessel v, the number of containers of all transshipments $t \in T$ with $v_t^1 = v$ was reduced by 1. [2]

The experiments reported in Section 4.2 were conducted on a Dell Precision T3620, Intel Core i7-6700 with 32GB RAM with the algorithm written in Java. For all the experiments the SA algorithm was allowed to run for 15 minutes. The distance penalty d^p for pairs of bollards which belong to different berthing zones and the safety distance d^s required between two BAs were set to 500 and $30m$, respectively. Finally, the container handling throughput p_c was set to 30 containers per hour for all cranes.

4.2 Experimental results

The first set of experiments utilized the iRace package to investigate the performance of the proposed constructive methods and to identify efficient parameter configurations for different instance characteristics. Instances were divided with respect to their quay size and number of berthing zones. This resulted in 6 groups which are each composed of 24 instances: S1, S2, M1, M2, L1 and L2. iRace explored the tuning space of the eight parameters whose respective ranges are provided in Table 2, while considering all 24 instances on a budget of 5000 runs per group. For the FF constructive method both the default TCTD evaluation and the delta approximation scheme were considered. The latter is denoted as FFd.

Table 3 reports the two best configurations indicated by iRace for each of the six groups of instances. These configurations were subsequently evaluated over the full set of 144 instances and for each instance the gap of their obtained TCTD from the best found TCTD was computed. The average value of their gap without considering infeasible solutions along with the number of infeasible solutions which each configuration

[2] All generated instances are publicly available at
`https://set.kuleuven.be/codes/datasets/TBACASPinstances/`

Parameter	Range	Parameter	Range
T_0	$10000 \rightarrow 500000$	$mcaf$	$0.01 \rightarrow 1$
cs	$10^{-6} \rightarrow 10^{-4}$	$maxNumRem$	$5 \rightarrow 20$
CM	{FF, BF, OBF, RCA, FFd}	u	{1, 2, 5, 10, 20} *
rcc	$0.01 \rightarrow 1$	stp	$0.1 \rightarrow 0.9$

Table 2 Examined parameters and their ranges.
*For unit u the value 1 was excluded for the large-size instances and value 20 was excluded for small and medium-size instances following preliminary experiments.

Sets	Configurations								Avg Gap (%)	# Inf
	T_0 ($\cdot 10^3$)	cs ($\cdot 10^{-6}$)	CM	rcc	$mcaf$	$maxNumRem$	u	stp		
S1	100	70	OBF	0.7	0.01	15	2	0.6	**2.75**	**11**
	60	70	OBF	0.4	0.01	15	2	0.4	**2.86**	**10**
S2	60	20	OBF	0.01	0.4	15	2	0.6	3.51	21
	150	20	BF	**0.01**	**0.01**	15	1	0.4	4.09	19
M1	250	40	BF	1.0	0.4	15	2	0.4	2.55	17
	100	40	BF	1.0	0.4	15	2	0.4	2.71	19
M2	100	10	OBF	0.7	0.7	10	1	0.4	2.59	14
	250	70	OBF	0.01	0.7	10	2	0.8	2.84	16
L1	200	90	BF	0.7	0.01	15	5	0.2	2.56	14
	200	70	OBF	0.7	0.01	15	5	0.2	2.49	13
L2	200	10	OBF	1.0	0.7	15	2	0.2	2.51	16
	200	40	FF	1.0	1.0	15	2	0.2	4.30	17

Table 3 Two best configurations indicated by iRace for each set of instances. The last two columns report each configuration's average gap from the best known solutions after they were applied to all the instances without considering infeasibilities and the number of infeasible solutions which they obtained, respectively.

obtained is also reported in Table 3. Finally, the configurations which obtained the best solutions for each instance during iRace's iterated racing procedure were also isolated. The best TCTDs along with the associated configurations' constructive method and minimum considered length unit u for the small, medium and large instances are reported in Tables 4, 5 and 6, respectively. For the majority of small-sized instances with $|V| = 10$, all BF, OBF and RCA constructive methods were included in the configurations which obtained the best solutions and therefore these instances are not reported in Table 4.

For small instances, Table 3 indicates how iRace suggests the OBF constructive method for both 1-zone and 2-zone settings. However, BF significantly outperforms OBF when it comes to congested S2 instances, providing the best solutions for all instances with $|V| = 25$, as shown in Table 4. Along with the fact that it works with the lowest values of u, rcc and $mcaf$ parameters, BF's dominance suggests that in these solutions the sets of compatible cranes impact the selection of individual BAs towards minimizing TCTD. The small quay size in combination with reduced number of possible berthing positions due to the quay discontinuity in S2 instances allows BF to reach the desired level of detail and further improve the solutions.

As problem size grows, results in Tables 3, 5 and 6 show that OBF is outperformed by BF for 1-zone settings in both medium and large instances, but not for 2-zone settings. This is due to the fact that for a given quay size and set of vessels, 1-zone

Instances				1-zone (S1)			2-zone (S2)		
$\|V\|$	$\|C\|$	$\|B_c\| \cdot u$	N_v^P	TCTD	CM	u	TCTD	CM	u
20	10	200	800	1635388	OBF	2	4841046	OBF	2
			1000	2123012	BF	2	6302266	OBF	2
		400	800	1648050	OBF	2	4820342	OBF	2
			1000	2119400	OBF	2	6244556	OBF, BF	2
	20	200	800	1638300	OBF	2	4823580	OBF	2
			1000	2119978	OBF	2	6255082	OBF	2
		400	800	1639650	OBF	2	4823580	OBF	2
			1000	2128354	OBF	2	6237344	OBF	2
25	10	200	800	2752854	OBF	1	7505453	**BF**	2
			1000	3482682	OBF	2	9693885	**BF**	1
		400	800	2779404	BF	2	7450672	**BF**	1
			1000	3523545	OBF	5	9534820	**BF**	1
	20	200	800	2734828	OBF	2	7495997	**BF**	1
			1000	3423370	OBF	2	9577967	**BF**	2
		400	800	2796202	FF	2	7450672	**BF**	1
			1000	3575880	BF	1	9573882	**BF**	1

Table 4 Best results found by iRace for small-sized instances with $|V| = 20$ and $|V| = 25$ along with the Constructive Method (CM) and minimum length unit (u) considered by the associated configuration

Instances				1-zone (M1)			2-zone (M2)		
$\|V\|$	$\|C\|$	$\|B_c\| \cdot u$	N_v^P	TCTD	CM	u	TCTD	CM	u
20	20	600	800	3042532	RCA	2	5962719	OBF	1
			1000	3893410	BF	1	7666780	OBF	5
		800	800	3055102	BF	2	6004032	OBF	2
			1000	3932838	BF	2	7570855	OBF	1
	30	600	800	3042640	RCA	2	6006240	OBF	5
			1000	3883221	BF	1	7562719	OBF	1
		800	800	3042072	BF	2	5939160	OBF	1
			1000	3894026	BF	2	7564629	OBF	1
25	20	600	800	5107236	BF	2	9911593	OBF	2
			1000	6591504	BF	2	12523087	OBF	2
		800	800	5133422	BF	2	9957970	OBF	5
			1000	6585770	OBF	5	12340555	OBF	1
	30	600	800	5118844	BF	2	9892584	OBF	1
			1000	6432285	BF	5	12357468	OBF	1
		800	800	5158034	BF	2	10059020	OBF	2
			1000	6440054	BF	2	12444235	BF	5
30	20	600	800	5844564	OBF	2	11173246	OBF	2
			1000	7251644	BF	2	14842583	OBF	2
		800	800	5905522	BF	2	11306800	OBF	1
			1000	7426535	BF	5	14323942	OBF	1
	30	600	800	5591722	BF	2	11155930	OBF	2
			1000	7521074	BF	5	14894068	OBF	2
		800	800	5758064	BF	2	11165463	OBF	2
			1000	7279025	BF	5	14764256	OBF	2

Table 5 Best results found by iRace for medium-sized instances along with the Constructive Method (CM) and minimum length unit (u) considered by the associated configuration

Instances				1-zone (L1)			2-zone (L2)		
$\|V\|$	$\|C\|$	$\|B_c\| \cdot u$	N_v^P	TCTD	CM	u	TCTD	CM	u
40	40	600	800	13183740	BF	5	19347761	FF	2
			1000	15600540	BF	5	23600372	OBF	2
		800	800	13397766	BF	2	19397001	FF	2
			1000	15792408	BF	2	23277417	**BF**	2
	60	600	800	13175740	OBF	5	19037980	FF	2
			1000	15361865	BF	5	23178830	FF	2
		800	800	13327910	BF	5	19655617	FF	2
			1000	15640530	OBF	5	22934142	**BF**	2
50	40	600	800	18273560	OBF	5	27428750	OBF	5
			1000	24264090	OBF	10	36230977	BF	2
		800	800	18680110	OBF	2	27813714	OBF	2
			1000	24753995	BF	5	37292649	**BF**	2
	60	600	800	18386460	BF	5	27102978	OBF	2
			1000	24523725	BF	5	36075190	OBF	5
		800	800	18685075	BF	5	28242673	OBF	2
			1000	24474740	OBF	2	37385585	**BF**	5
60	40	600	800	22767100	BF	5	32877065	OBF	2
			1000	30298895	BF	5	46323537	OBF	2
		800	800	23470285	OBF	5	32856987	FF	2
			1000	32873400	OBF	5	44835400	**BF**	2
	60	600	800	22960650	OBF	10	32451234	BF	2
			1000	31556760	OBF	5	44974185	BF	5
		800	800	23070716	BF	**2**	33580208	OBF	2
			1000	33623262	BF	**2**	45434710	**BF**	2

Table 6 Best results found by iRace for large-sized instances along with the Constructive Method (CM) and minimum length unit (u) considered by the associated configuration

settings offer a greater number of possible crane assignments for each BA, in addition to longer overlapping crane ranges. Therefore, the possibility of incompatible crane assignments in those settings is high and the requirement of conflict checks becomes more crucial. This is also confirmed by the reduced values of the $mcaf$ parameter in the configurations suggested by iRace for 1-zone settings, as reported in Table 3. BF's efficient handling of additional crane conflict possibilities is further highlighted by its behaviour for congested L2 instances with increased numbers of cranes and crane ranges, as reported in Table 6. Although it was not indicated by iRace as an efficient CM for this group, BF provided the best solutions for all L2 instances with $|C| = 60$, $|B_c| \cdot u = 800$ and $N_v^P = 1000$.

However, BF requires lengthier runtimes to converge and thus its contribution becomes limited for L1 settings, as indicated by the increased number of best solutions provided by OBF in Table 6 compared to M1 in Table 5. Tables 3 and 6 also indicate that in L2 settings with lower levels of congestion and reduced container loads, FF performs well when combined with high values for the rcc and $mcaf$ parameters, being able to quickly explore the large, but not highly constrained, search space. Finally, RCA provides the best solutions for only two of the least congested M2 instances and along with FFd they demonstrate poor performance across all instances.

The results presented in Tables 3-6 show that OBF generally performs better than BF overall and requires less tuning effort. The high infeasibility rates in Table 3 illustrate that the configuration of SA parameters is critical for finding feasible berth allocations in congested settings. The two configurations which were suggested by iRace for S1 instances, which utilize OBF constructive method, produce low average gaps

and numbers of infeasibilities and are therefore most suited for general purpose tuning. However, the number of infeasible solutions, the behaviour of the BF and OBF constructive methods in different settings and the diversity of various parameter values across the effective configurations highlight the significance of instance-specific tuning when it comes to the underlying problem.

Regarding the performance of the spatio-temporal neighbourhood operator, the values of the *stp* parameter in Table 3 indicate that it is crucial for improving the solutions in small and medium-sized instances. In larger settings with increased numbers of vessels, the contribution of the spatio-temporal operator is potentially cancelled out due to its effect upon computation time, which does not allow for fast convergence.

Concerning the effect of the discretization policy, the results demonstrate how considering high granularity is capable of improving the solutions across all settings. As expected, configurations with $u = 1$ and $u = 2$ perform better for the small-sized and M2 instances. $u = 2$ also results in better solutions for M1 and L2 instances. In L1 settings, where the requirement for fast convergence is more critical, $u = 5$ generally performs better. However, configurations with $u = 2$ provided best solutions for various L1 instances, including two of the most congested ones. The poor performance of configurations with $u = 10$ and $u = 20$, even when the search space is very large, proves that the obtained solutions are critically affected when significant numbers of possible berthing positions are disregarded.

The final set of experiments attempts to better evaluate the performance of the proposed approximation scheme. The second configuration which was suggested by iRace for L2 instances and which utilizes the FF constructive method, was also evaluated with the delta approximation scheme (FFd) over all instances. Table 7 reports the TCTDs and computation times for the 21 instances for which FFd's solution was obtained quicker than FF and its respective gap was less than 5%.

Table 7 indicates how FFd performs significantly better for medium-sized instances, with 17 of these 21 results being from this group. Although there are instances for which it presents competitive results or even outperforms FF in shorter computation time, its overall performance does not justify its usage over the conventional objective function calculation for the given time allowance. Nevertheless, the proposed approximation scheme demonstrates significant potential for use in more demanding dynamic environments.

5 Conclusions

The ever-increasing logistic pressures terminals are subject to when it comes to efficiently handling container transshipments requires a transition away from manual handling to algorithmic approaches. To this end, the present paper introduced the TBACASP, which incorporates Berth Allocation and Specific Quay Crane Assignment decisions with the aim of minimizing transshipment-related yard costs within a port terminal. An integrated formulation was presented to accurately model the layout of a typical terminal setting in addition to numerous real-world attributes for its components. Based on the developed formulation, an R&R heuristic was proposed to address the problem and provide good-quality solutions in short computation times.

An integrated set of benchmark instances for the TBACASP was made publicly available. An extensive experimental evaluation methodology was applied to investigate

Instances					FF		FFd		
Set	$\|V\|$	$\|C\|$	$\|B_c\| \cdot u$	N_v^P	TCTD	Time(s)	TCTD	Time(s)	Gap (%)
M1	20	20	800	800	3108176	730.255	3171332	426.333	1,99
M1	20	20	800	1000	4067148	714.396	4144528	645.802	1,87
M1	20	30	600	1000	4059804	899.920	4125278	**243.329**	1,59
M1	25	20	800	1000	7079664	856.944	7095000	513.052	0,22
M1	25	30	600	800	5230462	893.230	5235406	**129.145**	0,09
M1	25	30	600	1000	7057338	782.475	**6954192**	582.114	-
M1	25	30	800	800	5464234	786.301	**5286910**	776.610	-
M1	25	30	800	1000	7015596	547.185	7073144	315.146	0,81
M1	30	20	800	800	6433192	880.983	**6398694**	879.611	-
M1	30	30	600	800	6868008	853.535	6876860	814.642	0,00
M1	30	30	800	800	6702116	839.613	**6583392**	558.792	-
M2	20	30	800	800	6107808	855.248	6140622	631.492	0,53
M2	25	20	800	1000	13298300	771.847	13597848	665.789	2,20
M2	25	30	600	1000	13264453	823.469	**13174553**	804.036	-
M2	25	30	800	800	10248447	878.403	10364365	832.596	1,12
M2	30	20	600	800	11544708	540.316	**11544708**	460.499	0,00
M2	30	20	800	1000	16710486	831.327	**16691432**	816.294	-
L1	60	60	800	800	24776696	761.534	25882738	597.853	4,27
L2	60	40	600	800	35988171	815.763	37131629	**47.838**	3,08
L2	60	40	800	800	33998745	742.363	34829595	**213.104**	2,39
L2	60	40	800	1000	46983930	889.838	48200267	595.121	2,52

Table 7 Comparison of FF with FFd. TCTDs, computation times and respective gaps for the 21 instances for which FFd was faster than FF and the gap was below 5%. When FFd obtained better solutions, a dash(-) is shown.

the behaviour of different QCAP incorporation strategies and discover effective parameter configurations with respect to diverse instance features. The presented methodology, which makes use of the iRace package, may also be applied to other industrial problems which require integrations of different components.

Experimental results demonstrate the significance of instance-based tuning for the underlying problem and indicate efficient configurations of the algorithm for identifiable setting features. They additionally highlight the contribution of incorporating specific crane assignment decisions towards minimizing container transfer distances in congested settings. Finally, they confirm that considering highly discretized quay settings is capable of significantly improving the solutions of BAP variants.

The proposed approach is intended for real-time decision support, providing solutions of TBACASP in short computation times. However, the present paper only considers the transshipment-related objective of the problem. Future research will incorporate short-term objectives such as yard occupancy, vessel serving times and crane usage. The presented tuning methodology should also be extended to include these additional problem attributes in an effective manner. Finally, an exact approach for solving small-size instances would also present a relevant contribution to the domain and enable accurate assessment of heuristic solutions.

Acknowledgments

Editorial consultation provided by Luke Connolly (KU Leuven).

References

C. Bierwirth, F. Meisel, A follow-up survey of berth allocation and quay crane scheduling problems in container terminals, European Journal of Operational Research 244 (2015) 675–689.

C. Bierwirth, F. Meisel, A survey of berth allocation and quay crane scheduling problems in container terminals, European Journal of Operational Research 202 (2010) 615–627.

G. Giallombardo, L. Moccia, M. Salani, I. Vacca, Modeling and solving the Tactical Berth Allocation Problem, Transportation Research Part B: Methodological 44 (2010) 232–245.

I. Vacca, M. Salani, M. Bierlaire, An Exact Algorithm for the Integrated Planning of Berth Allocation and Quay Crane Assignment, Transportation Science 47 (2013) 148–161.

E. Lalla-Ruiz, J. L. González-Velarde, B. Melián-Batista, J. M. Moreno-Vega, Biased random key genetic algorithm for the Tactical Berth Allocation Problem, Applied Soft Computing Journal 22 (2014) 60–76.

M. Liu, C. Y. Lee, Z. Zhang, C. Chu, Bi-objective optimization for the container terminal integrated planning, Transportation Research Part B: Methodological 93 (2016) 720–749.

A. Imai, H. C. Chen, E. Nishimura, S. Papadimitriou, The simultaneous berth and quay crane allocation problem, Transportation Research Part E: Logistics and Transportation Review 44 (2008) 900–920.

Y. B. Türkoğullari, Z. C. Taşkin, N. Aras, I. K. Altinel, Optimal berth allocation, time-variant quay crane assignment and scheduling with crane setups in container terminals, European Journal of Operational Research 254 (2016) 985–1001.

Ç. Iris, D. Pacino, S. Ropke, A. Larsen, Integrated berth allocation and quay crane assignment problem: Set partitioning models and computational results, Transportation Research Part E: Logistics and Transportation Review 81 (2015) 75–97.

F. Meisel, C. Bierwirth, A Framework for Integrated Berth Allocation and Crane Operations Planning in Seaport Container Terminals, Transportation Science 47 (2013) 131–147.

O. Abou Kasm, A. Diabat, T. C. Cheng, The integrated berth allocation, quay crane assignment and scheduling problem: mathematical formulations and a case study, Annals of Operations Research (2019).

R. Bellio, S. Ceschia, L. Di Gaspero, A. Schaerf, T. Urli, Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem, Computers and Operations Research 65 (2016) 83–92.

H. G. Santos, T. A. Toffolo, C. L. Silva, G. Vanden Berghe, Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem, International Transactions in Operational Research 26 (2019) 707–724.

S. Chand, M. Wagner, Fast heuristics for the multiple traveling thieves problem, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, pp. 293–300.

T. Messelis, P. De Causmaecker, An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem, European Journal of Operational Research 233 (2014) 511–528.

F. Li, J.-B. Sheu, Z.-Y. Gao, Solving the continuous berth allocation and specific quay crane assignment problems with quay crane coverage range, Transportation Science

49 (2015) 968–989.

M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

Y. M. Park, K. H. Kim, A scheduling method for Berth and Quay cranes, Container Terminals and Automated Transport Systems: Logistics Control Issues and Quantitative Decision Support (2005) 159–181.

F. Meisel, C. Bierwirth, Heuristics for the integration of crane productivity in the berth allocation problem, Transportation Research Part E: Logistics and Transportation Review 45 (2009) 196–209.

A Appendix: Algorithms

Algorithm 1 First-Fit (FF)

Input: A current solution S

for all $v \in V_U^S$ **do**
  for all $a \in A_v$ **do**
    if $noconflicts(a, A^S)$ **then**
      $AssignCranes(a)$
      if $Covered(a, R_v^S)$ **then**
        $A_v^S \leftarrow a$
        Remove v from V^U
        break
      end if
    end if
  end for
end for

Algorithm 2 Best-Fit (BF)

Input: A current solution S

for all $v \in V_U^S$ **do**
  $Best, BestR \leftarrow \emptyset$
  $BestCost \leftarrow \infty$
  for all $a \in A_v$ **do**
    if $noconflicts(a, A^S)$ **then**
      $AssignCranes(a)$
      if $Covered(a, R_v^S)$ **then**
        if $Cost(a, A^S) < BestCost$ **then**
          $Best \leftarrow a$
          $BestCost \leftarrow Cost(a, A^S)$
          $BestR \leftarrow R_v^S$
        end if
      end if
    end if
  end for
  if $Best \neq \emptyset$ **then**
    $A_v^S \leftarrow Best$
    $R_v^S \leftarrow BestR$
    Remove v from V^U
  end if
end for

Algorithm 3 Check-Only Best-Fit (OBF)

Input: A current solution S

for all $v \in V_U^S$ **do**
  $Best, BestR \leftarrow \emptyset$
  $BestCost \leftarrow \infty$
  for all $a \in A_v$ **do**
    if $noconflicts(a, A^S)$ **then**
      if $Cost(a, A^S) < BestCost$ **then**
        $AssignCranes(a)$
        if $Covered(a, R_v^S)$ **then**
          $Best \leftarrow a$
          $BestCost \leftarrow Cost(a, A^S)$
          $BestR \leftarrow R_v^S$
        end if
      end if
    end if
  end for
  if $Best \neq \emptyset$ **then**
    $A_v^S \leftarrow Best$
    $R_v^S \leftarrow BestR$
    Remove v from V^U
  end if
end for

Algorithm 4 Redo Crane Assignments (RCA)

Input: A current solution S

for all $v \in V_U^S$ **do**
  for all $a \in A_v$ **do**
    if $noconflicts(a, A^S)$ **then**
      if $Cost(a, A^S) < BestCost$ **then**
        $Best \leftarrow a$
        $BestCost \leftarrow Cost(a, A^S)$
      end if
    end if
  end for
  if $Best \neq \emptyset$ **then**
    $A_v^S \leftarrow Best$
    Remove v from V^U
  end if
end for
for all $a \in A^S$ **do**
  $AssignCranes(a)$
  if $notCovered(a, R_v^S)$ **then**
    $A_v^S \leftarrow \emptyset$
    $R_v^S \leftarrow \emptyset$
    $V^U \leftarrow V^U \cup v_a$
  end if
end for

Algorithm 5 *AssignCranes*(a)

Input: A current solution S and a BA $a \in A^S$, with $v_a = v$

$R^S_{OLD} \leftarrow R^S_v$
$R^S_v \leftarrow \emptyset$
Sort C_a in descending order of $\frac{|B_c \cap B_a|}{|B_a|}$
for all $c \in C_a$ **do**
  $maxInterval \leftarrow [start_v, end_v]$
  for all $r \in (R^S_1, \cup ... R^S_{|V|}), : c_r \in C^N_c \cup c$ **do**
    if $Incompatible(r, < a, c, maxInterval^-, maxInterval^+ >)$ **then**
      $maxInterval \leftarrow maxInterval \setminus maxInterval \cap [start_r, end_r]$
      if $getLongestContinuousInterval(maxInterval) < mcaf \cdot (start_v - end_v)$ **then**
        break
      end if
    end if
  end for
  if $getLongestContinuousInterval(maxInterval) >= mcaf \cdot (end_v - start_v)$ **then**
    $maxInterval \leftarrow getLongestContinuousInterval(maxInterval)$
    $R^S_v \leftarrow R^S_v \cup < a, c, maxInterval^-, maxInterval^+ >$
    if $Covered(a, R^S_v) \vee (|R^S_v| = maxNumCranes_v)$ **then**
      break
    end if
  end if
end for

Algorithm 6 *Simulated annealing*

Input: Initial solution S_0, initial temperature T_0, cooling rate cs

$S \leftarrow S_0$
$S^* \leftarrow S_0$
$T \leftarrow T_0$
$dT \leftarrow cs * T_0$
while time limit not exceeded **do**
  Get random neighbour S' of S
  $\Delta \leftarrow TCTD(S') - TCTD(S)$
  if $\Delta < 0$ **then**
    $S \leftarrow S'$
    if $TCTD(S') < TCTD(S^*)$ **then**
      $S^* \leftarrow S$
    end if
  else
    Get random $p \in [0, 1]$
    if $p < e^{-\Delta/T}$ **then**
      $S \leftarrow S'$
    end if
  end if
  $T \leftarrow T - dT$
  if $T < 10^{-3}$ **then**
    $T \leftarrow T_0$
  end if
end while
return S^*

MISTA 2019

Water Wave Optimization for Integrated Bank Customer Service Scheduling

Xue-Qin Lu · Chen-Xin Wu · Min-Xia Zhang · Yu-Jun Zheng

Abstract In modern banks, efficient and quality customer services play a crucial role in the business success. Normally, there are a large number of customers to be served, but the service resources (in particular the number of customer service representatives) can be very limited. Therefore, bank customer service scheduling is a challenging problem, for which traditional exact optimization algorithms are often inefficient. In this paper, we present a problem of scheduling customer service representatives in different business teams to minimize the total loss, which consists of two parts: (1) the loss estimated based on the waiting time of customers and (2) the cost pay for the service representatives. This scheduling problem is modelled as an integer programming problem. To solve this problem, we propose an efficient evolutionary algorithm based on the water wave optimization meta-heuristic, where the propagation operator and refraction operator are elaborately adapted according to the solution space of this problem. The computational experiments demonstrate that the adapted WWO algorithm outperforms some state-of-the-art evolutionary algorithms in solving a set of test problem instances.

1 Introduction

For a commercial bank, customer service is a critical component of customer loyalty and hence revenue generation. Every day there are many customers who seek advices through bank website, APP, WeChat, call center, or other accessing channels, but the service resources, in particular the customer service representatives may be very limited. In order to provide fast and convenient service to customers with limited resources, many commercial banks have invested millions to deploy network-based self-services at their customer service centers. Such self-service in deed plays an important role because it can response quickly and provide 7 x 24 hours continuous

Xue-Qin Lu and Min-Xia Zhang
College of Computer Science & Technology, Zhejiang University of Technology, Hangzhou 310023, China
E-mail: luxueqin@nbufe.edu.cn, zmx@zjut.edu.cn
Chen-Xin Wu and Yu-Jun Zheng
Institute of Service Engineering, Hangzhou Normal University, Hangzhou 311121, China
E-mail: yujun.zheng@computer.org

services. When customers come and ask some relatively simple and repetitive questions, the self-service can replace service representatives to give advices or present what she/he wants quickly. Therefore, it can help decentralise the customer flow and save the service representatives resources dramatically. However, sometimes, such self-service can't present the direct answers, even expose some ambiguous information that may create uncertainty leading customers to seek more information and hence turn to the service representatives for advices. Or, customers ask some questions about sales guide or complex interaction scenarios, self-service has difficulty in responding such questions. Even under emergency situations, customers have to contact service representatives directly. To provide good service to customers in such cases, more serivce representatives need to work at the customer service centers. With the increasing employment costs, customer service managers, who are more concerned with the efficiency of service representatives and cost [1], have been facing increasing pressure in the balance between the employment costs and maintaining acceptable service efficiency. How to schedule the customer service representatives to meet the requirement is a challenging problem for customer service managers.

Usually, there are various business properties of questions from customers, such as financing business, credit card business, loan business, personal banking business and so on. With the diversify of communication ways, the customers may seek advices for their problems through different accessing channels. According to business properties or accessing channels, these questions can be categorized into some different teams. The service representatives are also divided into the corresponding teams according to the skills they are good at. The important factor for scheduling service representatives lies in the number of the customers in each team during some period. In real-world banks, customer service representatives scheduling problems are much more complicated, there are many aspects that should be taken into consideration, such as customers' arriving time, their patience in waiting, representatives' numbers, the weight of different teams and so on. Anyway, the managers try to determine the most economically optimal representatives' schedule to achieve a good trade-off between the maximization of service efficiency and minimization of customer service representatives.

In this paper, based on statistical customers historical data, and integrating some aspects in the process of real customer service, we first develop a mathematical model to formulate the customer service representatives scheduling problem. From the view of customer service managers to minimize the total cost and to maintain service quality, in the model we employ the unit loss cost estimated based on the customer waiting time and the unit payment cost for the customer service representatives as measurement, and set the maximum number of the service representatives into the mathematical model as constraints that any feasible service representatives schedule must satisfy. Some other typical examples of customer service-condition constraints are (i) the service representatives are divided into several teams and independent of each other (ii) the longest customer waiting time is less than certain time unit, otherwise the customer will leave. Once a customer abandons a visit, the bank loses more potential revenue, and (iii) the average working time in one day is less than or equal to eight hours, otherwise the bank managers should double or triple representatives' payment, etc. These service-condition constraints are also incorporated when developing models and solution methods. So our presented problem involves determining the customer service representatives schedule to minimize the waiting time and costs subject to the service-condition constraints.

The main contributions of this paper are two-fold:

- We present a customer service representative scheduling problem that considers both the loss caused by the waiting of customers and the cost pay for the service representatives.
- We propose an improved water wave optimization (WWO) algorithm with adapted operators to efficiently solve this problem.

In the remainder of this paper, we discuss some related work in Section 2 and formulate the problem in Section 3; Section 4 proposes an adapted WWO algorithm for the problem, Section 5 presents the computational experiment, and Section 6 concludes.

2 Related Work

In the earlier study, there existed a copious literature dealing with the problem of long-term staffing to meet time-varying demand just to improve service quality. Jennings et al. [2] developed an approximate procedure based on a time-dependent normal distribution so that the probability of a delay fall just below a target probability at all times. Ward Whitt [4] used infinite-server queueing models and analysis methods by predicting remaining call holding times to facilitate dynamic customer service representatives in a telephone call center with the objective of immediately answering all calls. These models didn't consider the cost because at that time the cost of service representatives were relatively low and the accessing channel was only telephone call. later, Bhandari et al. [3] concerned service-level constraints and provided a way to dynamically adjust the customer service representatives, in which permanent operators and temporary operators were employed. The author modelled the problem as a constrained Markov decision process and seek the optimal non-randomized policy by using the modified balance equations disjunctive constraints (MBEDC) algorithm, but its aim is to generate managerial insights into the effects of temporary operators. Miao et al. [16] studied the model of predicting and announcing delays and determine the most economically optimal anticipated delay so that the manages can control the trade-off between revenue loss and revenue from satisfaction. Owais et al. [6] categorized the call centres categorized in two types and through analysis their focuses to increase the number of sales (revenue) besides decreasing the expenses by smart utilization of data and the workforce. Different researchers focus on different perspectives and formulate models from different views, but the aims are the same, all in order to provide better service to customers. In this paper, we consider the customer service representatives scheduling problem with the objective of balancing the cost and service quality.

In different models there are different approaches to solve the problems, for our proposed models, because its decision variables are integers, it can be considered as a variant of the well-known integer programming (IP) problem, which is to find a set of best solutions in a continuous integers solution space to optimize the objective function. For integer programming problem, traditional deterministic algorithms have definite methods and relations when searching from one solution to another, and their results are the unique best solutions. For small-scale integer programming problem, they are better, but when the problems size become relatively large, they

easily become computationally intractable. In recent two decades, heuristic methods, in particular evolutionary algorithms inspired by natural biological systems, have been popular in solving such problems.Those methods do not guarantee finding the exact optimal solution in a single simulation run, but in most cases they are capable of finding acceptable solutions in a reasonable computational time.

Evolutionary algorithms, most of which were initially proposed for solving continuous optimization problems, cannot be directly applied to integer programming optimization, because their operators cannot guarantee that new solutions are still integers. Sakawa [8] conducted an early study on the application of genetic algorithm (GA) to integer programming, in which double strings using reference solutions with updating procedure are applied in the algorithm so that individuals are decoded to the corresponding feasible solution for integer programming problems. The results shows that the GA method has better feasibility and effectiveness than the branch and bound method in several numerical examples. There are also some other literature that using GA or its variants to solve the IP problem, and the results show the performance is better than traditional deterministic algorithms.

Particle swarm optimization (PSO) [9] is a heuristic method that models social behaviour of bird flocking to guide a swarm of solutions (called particles) towards the most promising regions of the search space. Parsopoulos et al. [5], [7] conducted an early study on the application of PSO to integer programming, and based on this, Kitayama [10] regarded integer programming problem as an optimization problem with discrete decision variables, and treat all decision variables as a continuous decision variable so that to treat discrete decision variables as a penalty function. Through mathematical and structural optimization problems, the results examines the validity of PSO for the mixed decision variables. There are other literature, in which the experiences results show the PSO has better global optimization ability and faster convergence speed in solving the IP problems.

Besides, there are other evolutionary algorithms to be employed in solving the IP problems, Omran [11] used two variants of differential evolution (DE) to solve the IP problems, and the results were even better than GA.

3 The Proposed Problem

In order to balance the cost and service quality, we model the customer service representatives scheduling problem based on the following two considerations: one is the cost pay for the service representatives, and the other is the loss cost estimated based on the waiting time of customers caused by poor service quality. The cost pay for the service representatives depends on the number of the representatives, the less representatives, the less cost. But less representatives will cause poor service quality and increase the loss cost. When a customer comes to service representatives for seeking advices, he will be assigned to an idle service representative, if there is no one idle, he has to wait. The longer he waits, the less his patience is, when his waiting time reaches the maximum value, he will give up waiting and leave. Each time a customer abandons a visit, the bank may lose revenue because the revenue is generated by serving customers. Different customers in different teams may cause different loss. So sometimes, the managers have to let more alternative service representatives to work to reduce the customer waiting time, even with double or triple payments to the alternative representatives. Therefore, the cost pay for the service representatives and

the loss lost are interact with each other and contain, the customer service managers tend to judge and weigh the payment for the representatives' service and the loss cost of the customer waiting time. In our model, we include the payment cost and the loss cost, and try to minimize the total cost.

Now we will give the formulation description of the problem. The input variables of our problem include:

- m, the number of different business teams in a bank customer service center.
- x_i, the customer service representatives number of team i.
- s_i, the time-average cost of paying for each regular service representatives of team i.
- β, the times of payment for alternative service representative to that for regular one, it is usually two or three times.
- ω_i, the importance weight of team i.
- ΔT, the maximum value of customer waiting time.
- l_i, the loss cost per unit time incurred for each customer waiting in group i.
- α_i, the penalty coefficient to indicate the loss rate when customer waiting time exceeds ΔT.
- $\hat{N}_i$, the number of all customers that representatives serve in group i during some period, such as eight hours, and it can be estimated according to historical data.
- t_{ij}^a, the arrival time of customer j in group i. In this paper, we suppose the customer arrive time is generated from uniformly distributed random.
- t_{ij}^c, the service time of customer j in group i. We assume the service time randomly distributes in $2 \sim 10$ minutes.
- T, the regular working period, such as eight hours a day, and forty or fifty-six hours a week.

The decision variables are an integer vector $\mathbf{x} = \{x_1, x_2, ..., x_i, ...x_m\}$, each dimension of which is denoted as the customer service representative number of group i. Suppose the maximum number of regular and alternative representatives in each team is C_{rmax} and C_{amax} respectively, the maximum number of x_i is $C_{rmax}+C_{amax}$.

Besides input variables and the decision variables, there are some important intermediate variables to be calculated. Here subscript $i(1 \leq i \leq m), j, k(1 \leq i \leq C_{rmax} + C_{amax})$ represent team, customer and service representative respectively.

- t_{ij}^s, the beginning time of the service for customer j in team i.
- τ_{ik}, the time when one customer service representative k in group i completes service for the current customer.
- q_i, the total number of customers whose waiting time exceeds ΔT in team i.
- T_{ci}, the total waiting time of all customers in team i.
- $\hat{T}_i$, the total waiting time-out in team i.
- f_l, the total loss cost based on the waiting time of the customers.
- f_p, the total payment cost to all service representatives.

At time $t = 0$, when the first customer comes, all service representatives are idle, so the waiting time of the first customer in each group is zero. We have

$$\tau_{i1} = 0 \tag{1}$$

$$t_{i1}^s = 0 \tag{2}$$

$$t_{i1}^w = 0 \tag{3}$$

When the first customer in each group i is completed serve, we have

$$\tau_{i1} = t_{i1}^{c} \tag{4}$$

At first, in each group the number of customers is less than the service representatives' number, we have

$$t_{ij}^{w} = 0 \tag{5}$$

When an customer service representatives k in group i complete serving customer j, the value of his τ_{ik} will be updated as follows:

$$\tau_{ik} = t_{ij}^{s} + t_{ij}^{c} \tag{6}$$

The customers come one by one, when customer j comes at time t_{ij}^{a}, it will wait if there is no idle service representatives or go to the customer service representatives who completes the previous customers at the earliest, so its started time and the waiting time are calculated as follows:

$$t_{ij}^{s} = \max(t_{ij}^{a}, \min_{1 \le k \le x_i}(\tau_{ik})) \tag{7}$$

$$t_{ij}^{w} = t_{ij}^{s} - t_{ij}^{a} \tag{8}$$

So the total waiting time of all customers can be calculated as follows:

$$T_{ci} = \sum_{j=1}^{\hat{N}_i} t_{ij}^{w} \tag{9}$$

When t_{ij}^{w} exceeds its maximum value ΔT, the customer will give up this visit. The total waiting time-out in team i can be calculated as follows:

$$\hat{T}_i = q_i \Delta T \tag{10}$$

Considered into the different teams may have different weights and loss rate, the total loss can be calculated as follows:

$$f_l = \sum_{i=1}^{m} \omega_i T_{ci} l_i + \sum_{i=1}^{m} \omega_i \hat{T}_i \alpha_i l_i \tag{11}$$

Besides the loss cost, in our model, we also include the cost to pay for the service representatives. If alternative representatives work, they will be paid more than regular service. So the total payment cost f_h is calculated as follows:

$$f_h = \begin{cases} \sum_{i=1}^{m} \omega_i x_i s_i T, & if \quad x_i \le C_{rmax} \\ \sum_{i=1}^{m} \omega_i C_{rmax} s_i T + \sum_{i=1}^{m} \omega_i (x_i - C_{rmax}) \beta s_i T, & else \end{cases} \tag{12}$$

So the total cost is:

$$F(\mathbf{x}) = f_l + f_h \tag{13}$$

Our aim is to find an optimal solution $\mathbf{x}^*$ to minimize the total cost of all business teams:

$$F(\mathbf{x}^*) = \min F(\mathbf{x}) \tag{14}$$

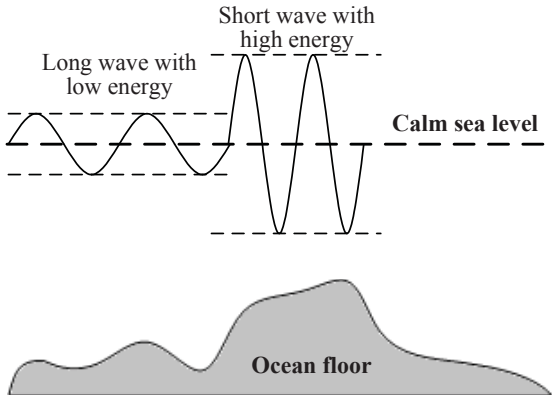


Fig. 1 Different wave shapes in deep and shallow water.

Because the decision variables $\mathbf{x}$ are integers, under the assumptions for the arrival process and service-time distributions, our proposes problem can be considered as a variant of the well-known integer programming (IP) problem. In some real large and medium-sized banks, the numbers of service representatives and the business teams are often large, traditional deterministic methods for IP problems may computationally intractable. In recent decades, evolutionary algorithms have become popular approaches for solving IP problems.

4 A Water Wave Optimization Algorithm for the Problem

Water wave optimization (WWO) is a relatively new evolutionary algorithm, which is inspired by the shallow water wave theory and developed by Zheng [13]. Because its control parameters are relatively few, and it is easy to perform, WWO is widely used to solve variable scheduling problems. For effectively solving our proposed problem of scheduling customer service representatives, we adapt the operators of the WWO algorithm. Before describing our algorithm in detail, we first give a brief introduction of WWO.

4.1 The Basic WWO Algorithm

In the original WWO algorithm, solution space is analogous to the seabed area, a solution $\mathbf{x}$ is analogous to a wave with height $h_{\mathbf{x}}$ and wavelength $\lambda_{\mathbf{x}}$, solution fitness is analogous to the wave height. The central to the algorithm is the wave movement, which indicates that the shorter the wave length is, the higher the energy, and thus the smaller the range the wave propagate (search), as illustrated by Fig. 1.

In a continuous search space, the propagation operation is done by shifting each dimension d of the solution as

$$x'_d = x_d + rand(-1, 1) \cdot \lambda_{\mathbf{x}} L_d \tag{15}$$

where *rand* is a function that generates a uniformly distributed random number within the specified range, and L_d is the length of the dth dimension of the search space.

All wavelength are initially set to 0.5 , and then updated based on fitness $f(\mathbf{x})$ after each generation as

$$\lambda_{\mathbf{x}} = \lambda_{\mathbf{x}} \alpha^{-(f(\mathbf{x})-f_{\min}+\varepsilon)/(f_{\max}-f_{\min}+\varepsilon)} \tag{16}$$

where $f_{\max}$ and $f_{\min}$ are the maximum and minimum fitness among the population, respectively, α is the wavelength reduction coefficient, and ε is a very small value to avoid division by zero.

Besides propagation, WWO is equipped with two other operators: refraction and breaking. All wave height are initially to a constant integer $h_{\max}$. If a wave has not been improves after one generation, its hight will be reduced by one to simulate the loss of energy. The refraction operator performs on any wave whose height decreases to zero to avoid search stagnation. It makes the stagnant wave learn from the current best wave $\mathbf{x}^*$ at each dimension d as:

$$x'_d = N(\frac{x^*_d + x_d}{2}, \frac{|x^*_d - x_d|}{2}) \tag{17}$$

where $N(\mu, \sigma)$ generates a Gaussian random number with mean μ and standard deviation σ. After refraction, its wavelength is updated according to the ratio of the new fitness to the original fitness as:

$$\lambda_{\mathbf{x}'} = \lambda \frac{f(\mathbf{x})}{f(\mathbf{x}')} \tag{18}$$

The breaking operator breaks a newly found best wave $\mathbf{x}^*$ into a series of solitary waves, each of which is obtained by randomly selecting k dimensions (where k is a random number between 1 and a predefined upper limit $k_{\max}$), and at each dimension d updating the component as:

$$x'_d = x^*_d + rand(0,1) \cdot \beta L_d \tag{19}$$

where β is the breaking coefficient. If the fittest one among the solitary waves is better than $\mathbf{x}^*$, it will replace $\mathbf{x}^*$ in the population.

Algorithm 1 presents the procedure of the basic WWO algorithm, which is originally proposed for continuous optimization problems. WWO has attracted considerable attention since it was proposed. There are various improved versions of WWO. Recently, Zheng et al. [21] propose a systematic approach that consists of a set of basic steps and strategies for adapting WWO to solve various combinatorial optimization problems.

4.2 Adapted Operators for the Considered Problem

WWO has been applied to solve many problems and demonstrated its capability, but it can't be used directly to solve the integrated bank customer service scheduling problem. To efficiently solve our proposed problem, we elaborately adapted the operators of WWO algorithm according to the solution space.

Algorithm 1 The basic WWO algorithm.

```
Randomly initialize a population of N solutions (waves);
while the termination condition is not satisfied do
    for each wave x ∈ P do
        Propagate x to a new x' according to Eq. (15);
        if f(x') > f(x) then
            Replace x with x';
            if f(x') > f(x*) then
                Break x' into new waves according to Eq. (19);
                Update x* with the best one among the new waves and x';
        else
            h_x ← h_x − 1;
            if h_x = 0 then
                Refract x to a new x' according to Eq. (17);
                Update λ_x' according to Eq. (18);
    Update the wavelengths of the waves according to Eq. (16);
return the best wave found so far.
```

First, we construct the fitness function $f(\mathbf{x})$ as :

$$f(\mathbf{x}) = \frac{(F_{\max} - F(\mathbf{x}) + \varepsilon)}{(F_{\max} - F_{\min} + \varepsilon)} \tag{20}$$

where F is the objective function, $F_{\max}$ and $F_{\min}$ are the maximum and minimum objective function values in the population, and ε is a small constant to avoid zero-division-error.

And then, we adapted the three operators of WWO algorithm to make it solve our problem well.

4.2.1 Propagation Operator

In our proposed problem, we regard the schedule of the customer service representatives in different teams as a whole, so its solution population consists of some sets of m non-negative integers. That is, each solution $\mathbf{x}$ is a set of representatives $\{x_1, x_2, ...x_m\}$ and its components x_i should be subject to the constraints of non-negative integer and its maximum value when propagating between solutions. Suppose a solution $\mathbf{x}$ is propagated to $\mathbf{x}'$ at dimension i, we can't simply use the Eq. (15) because that will cause the component x_i to be a real number and may cause the solution out of boundary or to be a negative value.

We first make the value of $rand(-1,1) \cdot \lambda_{\mathbf{x}} L_d$ in Eq.15 to be an integer. As there's a high probability that this value will be near zero, here we adopt such a strategy: if $rand(-1,1) \cdot \lambda_{\mathbf{x}} L_d$ is negative, we round it down to be integer; if it is positive, we round it up to be integer. The aim to adopt this strategy is to make the numbers near zero ,that is the numbers in $[-1,1]$, far away zero and to be -1 or 1, which can avoid the propagation to be at a standstill. So the new propagation operator is adapted as:

$$x'_d = \begin{cases} x_d + ceil(rand(-1,1) \cdot \lambda_{\mathbf{x}} L_d), & if \quad rand(-1,1) \cdot \lambda_{\mathbf{x}} L_d > 0 \\ x_d + floor(rand(-1,1) \cdot \lambda_{\mathbf{x}} L_d), & else \end{cases} \tag{21}$$

where *ceil* is a function that round a real number up to an integer, and *floor* is a function that round a real number down to an integer.

After the propagation, if some component of the solution $\mathbf{x}'$ is negative or beyond its maximum value, both of these are out of boundary. In this case, we set it to be a random number within the range.

4.2.2 Refraction Operator

In the original WWO algorithm, when a wave height decreases to zero, the refraction operator is performed by generating Gaussian random number to avoid search stagnation, shown as Eq.17. For an integer programming like our proposed problem, when employing Gaussian random number to perform the refraction, the number needs to be rounded into an integer, which will cause duplicate values and degrade the algorithm performance. So we renovate the operator to be adaptive our problem. The essence of the refraction operator is to reset the solution and make the components of the solution to learn from the current best wave $\mathbf{x}^*$. In our adapted refraction, we first select some random dimensions of the solution instead of all dimensions, and make the components in these dimensions to learn from the current best solution by add or subtract a random value between itself and the best solution at the same dimension. For each selected dimension d, the new refraction is adapted as:

$$x'_d = x_d \pm randint(0, |x^*_d - x_d|) \tag{22}$$

where *randint* is a function that generates a random integer between the specified range.

Also, some components of the solution may be negative or beyond its maximum value, we employ the same way like in propagation operator: set the value of the components out of boundary to be a random number within the range.

After refraction, each solution wavelength $\lambda_{\mathbf{x}}$ is updated according to the new fitness, showed as:

$$\lambda_{\mathbf{x}} = \frac{(f_{\max} - f_{\mathbf{x}} + \varepsilon)}{(f_{\max} - f_{\min} + \varepsilon)} \tag{23}$$

4.2.3 Breaking Operator

In the original WWO, the breaking operator is used to create some new solutions near the found best solution, in essence, it is a local search. Researches have shown that performing local exploitation around some promising solutions can greatly facilitate the evolution of population based heuristic algorithms [14,15]. In our adapted breaking operator, we use a simple local search strategy, in which the operator randomly selects k dimensions (where k is a random number between 1 and a predefined upper limit $k_{\max}$), and at each selected dimension d updating the component as:

$$x'_d = x^*_d + randint(-1, 1) \tag{24}$$

Because the smallest distance between any two integer positions is one, we adopt one as an unit step size of the local search, that is, the solution randomly goes forward or backward by a unit step, or unchanged to do local search. In our problem, such a strategy can make the local search more exactly, and improve the local exploiting ability. After the local search, the component out of boundary will also be set a random number within the range. If the new solution is better than $\mathbf{x}^*$, it will replace $\mathbf{x}^*$ in the population.

4.2.4 The Algorithm framework

Algorithm 2 presents the procedure of the adapted WWO algorithm. Suppose the population size is N , then each iteration requires $N + k_{max}/2$ fitness evaluation times (NFE) in average.

Algorithm 2 The Adapted WWO Algorithm.

```
Randomly initialize a population of N solutions (waves);
Calculate the fitness of every solution according to Eq.20
while the termination condition is not satisfied do
    for each wave x ∈ P do
        Propagate x to a new x' according to Eq. (21);
        if f(x') > f(x) then
            Replace x with x';
            if f(x') > f(x*) then
                Break x' into a new wave according to Eq. (24);
                if new is better than x* then
                    Update x* with the new solution;
        else
            h_x ← h_x − 1;
            if h_x = 0 then
                Refract x to a new x' according to Eq. (22);
                Update λ_x' according to Eq. (23);
    Update the wavelengths of the waves according to Eq. (23);
return the best wave found so far.
```

5 Computational Experiments

5.1 Experimental Instances

In order to verify the performance of the adapted algorithm, a set of 15 customer service representatives schedule instances in banks are considered for our experiments. The test suite contains 10 small- and medium- size instances (#1 $\sim$ #10) and 5 large-size instances (#11 $\sim$ #15), all of which are constructed based on the data from different sizes of 15 banks in different areas in China, their features are as summarized in Table 1.

On the test instances, we compare the adapted WWO algorithm with the following five meta-heuristic methods:

- The genetic algotithm (GA) with local search, denoted by GA_LS [19].
- The comprehensive learning particle swarm optimization, denoted by CLPSO [12].
- The enhanced fireworks algorithm (EFWA) [20].
- The biogeography-based algorithm (BBO) [17].
- The differential evolution (DE) algorithm [18].

Table 1 Summary of the test instances

#	m	C_{rmax}	C_{amax}	$\hat{N}_i$	$MNFE$
1	3	10	2	$2400 \sim 3500$	500
2	5	15	3	$3600 \sim 5200$	800
3	6	20	3	$5000 \sim 7500$	1000
4	8	27	4	$6500 \sim 10000$	1200
5	9	31	5	$6000 \sim 11000$	1300
6	10	32	5	$6600 \sim 12000$	1400
7	12	38	6	$7500 \sim 14000$	1600
8	15	40	7	$8000 \sim 15000$	2000
9	16	50	8	$8200 \sim 18000$	2100
10	18	55	9	$8500 \sim 19000$	2200
11	20	60	10	$9000 \sim 21000$	2400
12	22	70	11	$1000 \sim 22000$	2600
13	24	75	12	$11000 \sim 24000$	2800
14	25	80	13	$12000 \sim 26000$	3000
15	30	90	15	$14000 \sim 30000$	3500

5.2 Experimental Parameter Setting

For WWO, we empirically set the population size to 10, the other parameters are set as sugested in the literature [13]. The parameters of the other five algorithms are set as suggested in the literatures [12],[19], [17], [20], [18], and we make the variable value into integer when evolving solutions. Note that we just adopt a fixed parameter setting for every algorithm to evaluate its overall performance on the whole suite rather than make any effort in finding the best parameter settings for every problem.

In the experiments, we set the maximum numbers of fitness evaluation times (MNFE) for different instances, shown in the last column of Table 1. Moreover, each test instance is performed over 30 independent runs by using each algorithm, and the evaluation is based on the average performance over the 30 runs.

5.3 Experimental Results

Table 2 shows the experimental results of the comparative algorithms on the test instances, which records the best, worst, median and standard deviation of the resulting objective function values among the 30 runs. The minimum best and median values among the six algorithms are in boldface. We have also conducted t-test on the running results between WWO and each of the other algorithms, and mark † before the median values in columns $3 \sim 7$ if WWO has statistically significant performance improvement over the corresponding algorithms (at 95% confidence level).

As we can see, on the first small-size instance #1, all the algorithms can obtain the best objective value, and except BBO and CLPSO, the others can obtain the best median value. On the second small-size instance #2, except BBO and CLPSO, the others can obtain the best objective value, GA_LS and EFWA can obtain the best median value. On the third small-size instance #3, except BBO and CLPSO, the others can obtain the best objective value, but only WWO can obtain the best median value which is much less than the other algorithms. On the remaining instances except #11, WWO not only always obtains the minimum median values but also uniquely obtains the minimum best values. Particularly, on the last four large-size

Table 2 Comparative results on the test instances

#	Metric	BBO	CLPSO	DE	EFWA	GA_LS	WWO
m1	Median	4.95E+03†	4.92E+03†	**4.81E+03**	**4.81E+03**	**4.81E+03**	**4.81E+03**
	Worst	5.28E+03	5.17E+03	4.89E+03	5.00E+03	4.92E+03	4.92E+03
	Best	**4.81E+03**	**4.81E+03**	**4.81E+03**	**4.81E+03**	**4.81E+03**	**4.81E+03**
	Std	1.30E+02	9.97E+01	2.01E+01	5.97E+01	3.13E+01	3.83E+01
m2	Median	1.15E+04†	1.09E+04†	1.05E+04	**1.04E+04**	**1.04E+04**	1.06E+04
	Worst	1.28E+04	1.19E+04	1.10E+04	1.08E+04	1.09E+04	1.09E+04
	Best	1.09E+04	1.04E+04	**1.04E+04**	**1.04E+04**	**1.04E+04**	**1.04E+04**
	Std	4.72E+02	3.01E+02	1.42E+02	1.06E+02	1.67E+02	1.32E+02
m3	Median	2.00E+04†	1.95E+04†	1.81E+04†	1.81E+04†	1.81E+04†	**1.79E+04**
	Worst	2.14E+04	2.07E+04	1.87E+04	1.93E+04	1.96E+04	1.85E+04
	Best	1.84E+04	1.84E+04	**1.78E+04**	**1.78E+04**	**1.78E+04**	**1.78E+04**
	Std	6.51E+02	5.14E+02	2.33E+02	3.64E+02	4.42E+02	1.79E+02
m4	Median	3.87E+04†	3.46E+04†	3.27E+04†	3.21E+04†	3.29E+04†	**3.19E+04**
	Worst	4.64E+04	3.62E+04	3.38E+04	3.46E+04	3.60E+04	3.33E+04
	Best	3.53E+04	3.21E+04	3.19E+04	3.12E+04	3.17E+04	**3.11E+04**
	Std	2.88E+03	1.15E+03	3.93E+02	7.76E+02	1.14E+03	5.37E+02
m5	Median	4.28E+04†	4.02E+04†	3.62E+04†	3.64E+04†	3.76E+04†	**3.52E+04**
	Worst	4.71E+04	4.34E+04	3.74E+04	4.10E+04	4.22E+04	3.85E+04
	Best	3.70E+04	3.72E+04	3.45E+04	3.37E+04	3.50E+04	**3.30E+04**
	Std	2.49E+03	1.48E+03	6.99E+02	1.61E+03	1.84E+03	1.46E+03
m6	Median	5.62E+04†	5.08E+04†	4.74E+04†	4.66E+04†	4.81E+04†	**4.47E+04**
	Worst	7.68E+04	5.51E+04	4.94E+04	5.12E+04	5.22E+04	4.84E+04
	Best	5.05E+04	4.80E+04	4.58E+04	4.41E+04	4.50E+04	**4.34E+04**
	Std	4.82E+03	1.65E+03	7.06E+02	1.87E+03	1.75E+03	1.44E+03
m7	Median	8.65E+04†	7.50E+04†	7.07E+04†	7.03E+04†	7.24E+04†	**6.67E+04**
	Worst	5.43E+05	8.34E+04	7.49E+04	7.69E+04	7.98E+04	7.57E+04
	Best	7.52E+04	7.19E+04	6.71E+04	6.46E+04	6.52E+04	**6.25E+04**
	Std	1.03E+05	2.60E+03	1.68E+03	2.94E+03	3.29E+03	2.89E+03
m8	Median	1.93E+05†	1.05E+05†	9.21E+04†	9.11E+04†	9.63E+04†	**8.66E+04**
	Worst	1.12E+06	1.11E+05	9.56E+04	1.80E+05	1.10E+05	9.68E+04
	Best	1.06E+05	9.03E+04	8.76E+04	8.61E+04	8.64E+04	**8.27E+04**
	Std	2.88E+05	4.69E+03	2.11E+03	2.10E+04	5.34E+03	3.79E+03
m9	Median	1.51E+05†	1.32E+05†	1.25E+05†	1.13E+05†	1.25E+05†	**1.11E+05**
	Worst	3.37E+05	1.39E+05	1.35E+05	1.31E+05	1.64E+05	1.23E+05
	Best	1.31E+05	1.20E+05	1.12E+05	1.06E+05	1.11E+05	**1.00E+05**
	Std	5.58E+04	3.58E+03	5.26E+03	6.10E+03	9.33E+03	5.74E+03
m10	Median	3.70E+05†	1.73E+05†	1.64E+05†	1.52E+05†	1.61E+05†	**1.47E+05**
	Worst	1.16E+06	1.82E+05	1.75E+05	1.67E+05	1.89E+05	1.68E+05
	Best	1.54E+05	1.58E+05	1.52E+05	1.39E+05	1.42E+05	**1.31E+05**
	Std	2.52E+05	5.75E+03	6.35E+03	5.93E+03	1.02E+04	8.20E+03
m11	Median	2.09E+05†	2.09E+05†	1.38E+05†	**1.34E+05**	1.43E+05†	1.35E+05
	Worst	2.33E+05	2.26E+05	1.42E+05	1.38E+05	1.60E+05	1.44E+05
	Best	1.83E+05	1.86E+05	1.35E+05	**1.33E+05**	1.35E+05	**1.33E+05**
	Std	9.34E+03	9.26E+03	1.70E+03	1.27E+03	5.51E+03	2.37E+03
m12	Median	4.00E+05†	2.38E+05†	2.15E+05 †	2.15E+05	2.40E+05†	**2.04E+05**
	Worst	1.32E+06	2.50E+05	2.28E+05	1.91E+06	3.06E+05	2.31E+05
	Best	2.54E+05	2.21E+05	1.96E+05	1.98E+05	2.16E+05	**1.78E+05**
	Std	2.76E+05	6.34E+03	6.45E+03	3.07E+05	1.99E+04	1.26E+04
m13	Median	5.56E+05†	3.04E+05†	2.61E+05†	2.55E+05	2.87E+05†	**2.51E+05**
	Worst	1.74E+06	3.21E+05	2.98E+05	2.83E+05	3.13E+05	3.03E+05
	Best	3.03E+05	2.71E+05	2.45E+05	2.40E+05	2.68E+05	**2.26E+05**
	Std	3.50E+05	1.16E+04	1.07E+04	1.06E+04	1.36E+04	1.89E+04
m14	Median	1.38E+06†	3.40E+05†	3.46E+05†	3.03E+05	3.34E+05†	**2.79E+05**
	Worst	2.33E+06	3.64E+05	5.25E+05	8.72E+05	4.29E+05	5.36E+05
	Best	3.47E+05	3.23E+05	2.95E+05	2.79E+05	2.82E+05	**2.54E+05**
	Std	5.70E+05	9.87E+03	6.47E+04	1.03E+05	2.75E+04	5.75E+04
m15	Median	1.56E+06†	4.60E+05	7.37E+05 †	3.99E+05	4.47E+05	**3.87E+05**
	Worst	2.82E+06	4.95E+05	1.74E+06	8.16E+05	1.03E+06	1.30E+06
	Best	4.65E+05	4.09E+05	3.98E+05	3.59E+05	4.03E+05	**3.44E+05**
	Std	7.48E+05	2.18E+04	3.35E+05	1.08E+05	1.53E+05	2.47E+05

instances, the results of WWO are much better than the other five algorithms. On the instance #11, although EFWA can obtain the best objective value, a little better than WWO, WWO can also obtain the minimum best value.

According to the statistical test results, we can see that on the small-size instances, there is no significantly differences among these algorithm. On medium-size instances, WWO perform significantly better than BBO, CLSPO, DE, EFWA and GA_LS. On large-size instances, there is no significantly differences among WWO, and EFWA, but they perform significantly better than BBO, CLPSO, DE and GA_LS. The larger the instance size, the more obvious the performance advantage of WWO over the other algorithms. That indicates WWO has more global search ability in exploring large solution spaces.

By comparing the convergence curves, we can see that at the early stage of the search, WWO converges faster than the others, and all algorithms converges their own best solutions at the later stage of the search, as illustrated Fig. 2. But WWO converges to a better solution than the others, this is because WWO can explore more areas in the early stage, and use the adapted breaking operator to improve the local search ability so that the algorithm has capability of jumping out of the local optima where the others may be trapped. Therefore, WWO has a better solution in most given test instances.

In summary, the experimental results show that the overall performance of WWO is the best one among the five comparative algorithms on our test instances, which demonstrates that the adapted WWO can effectively solve our proposed problem, its solution can provide banks managers for reference.

6 Conclusion

In this paper, we first construct a model of customer service representatives scheduling at customer service in banks from the view of cost, and employ the WWO algorithm to solve this problem. WWO is a relatively new metaheuristic optimization method and show good performance on many scheduling problems. Compared with other algorithms on some instances, the adapted WWO algoritm show its advantage and can obtain the optimal solution for this problem.

Our ongoing work will consider other related factors that effect the cost and improve the model to make it more similar to real situation, so that to provide advices for banks managers. It is also desirable that adjusts the operators of the WWO algorithm to adopt our model, especially in local search.

References

1. Gans, N., Koole, G., Mandelbaum, A.: Telephone call centers: tutorial, review, and research prospects. Manufacturing & Service Operations Management 5(2), 79-141 (2003).
2. Jennings, O.B., Mandelbaum, A., Massey, W.A., Whitt, W.: Server staffing to meet time-varying demand. Management Science 42(10), 1383-1394 (1996).
3. Bhandari, A., Scheller-Wolf, A., Harchol-Balter, M.: An exact and efficient algorithm for the constrained dynamic operator staffing problem for call centers. Management Science 54(2), 339-353 (2008).
4. Whitt, W.: Dynamic staffing in a telephone call center aiming to immediately answer all calls. Operations Research Letters 24(5), 205 -212(1999).

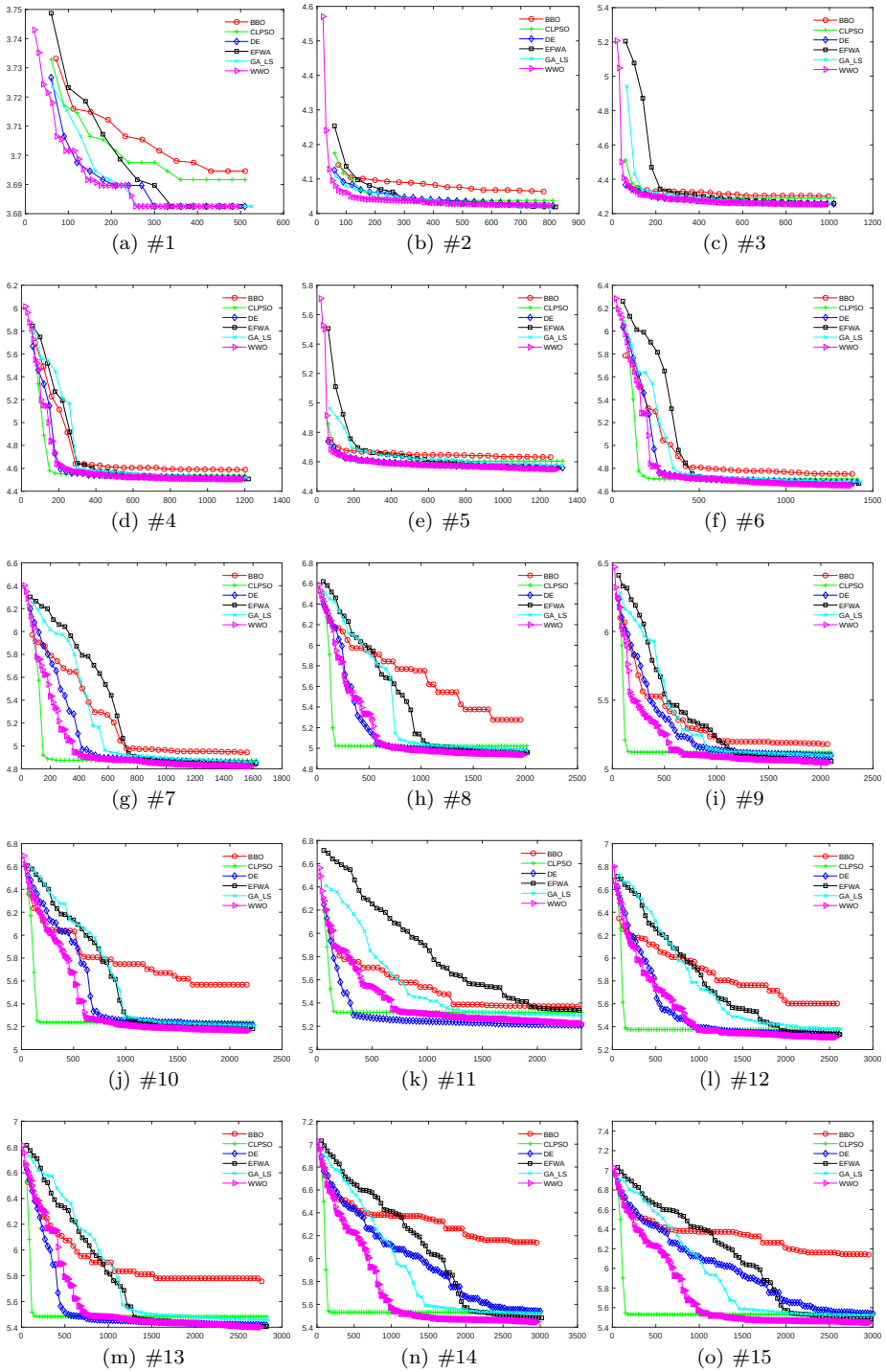


Fig. 2 Convergence curves of the algorithms on the test instances. The x-axis denotes NFEs, and the y-axis denotes the base-10 logarithm of the median values.

5. Parsopoulos, K.E., M.V.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing 1(2-3), 235-306 (2002).
6. Owais, R., Ali, M. Q., Sharifullah, K., Sadaf,A.: Intelligent decision making and planning for call center. In: 2019 International Conference on Computer and Information Sciences(ICCIS) (2019).
7. Laskari, E.C., Parsopoulos, K.E., Vrahatis, M.N.: Particle swarm optimization for integer programming. In: Congress on Evolutionary Computation (2002).
8. Sakawa, M., Kato, K.: Integer programming through genetic algorithms with double strings based on reference solution updating. In: Conference of the IEEE Industrial Electronics Society (2000).
9. Kennedy, J.: Particle Swarm Optimization. Springer US. 760-766(2010).
10. Kitayama, S., Yasuda, K.: A method for mixed integer programming problems by particle swarm optimization. Electrical Engineering in Japan 157(2), 40-49 (2006).
11. Omran, M.G.H., Engelbrecht, A.P.: Differential evolution for integer programming problems. In: IEEE Congress on Evolutionary Computation (2007)
12. Liang, J.J., Qin, A.K., Suganthan, P.N., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation 10(3), 281-295 (2006)
13. Zheng, Y.J.: Water Wave Optimization: a new nature-inspired metaheuristic. Computers & Operations Research. 55, 1-11 (2015).
14. Krasnogor, N., Smith, J.: A memetic algorithm with self-adaptive local search: TSP as a case study. In: Proceedings of 2nd Annual Conference on Genetic and Evolutionary Computation. 987-994 (2000)
15. Liu, B., Wang, L., Jin, Y.H.: An effective pso-based memetic algorithm for flow shop scheduling. In: IEEE Transactions on Systems, Man, and Cybernetics. 37, 18-27 (2007).
16. Yu, M., Chang, C., Zhao, Y., Liu, Y.: Announcing delay information to improve service in a call center with repeat customers. In: IEEE Access. 7, 66281-66291 (2019).
17. Simon, D.: Biogeography-based optimization. IEEE Transactions on Evolutionary Computation. 12(6), 702–713 (2008).
18. Storn R, Price K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization. 11(4), 341-359 (1997).
19. Goncalves, J., Mendes, J., Resende,M.: A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research. 167, 77–95 (2005).
20. Zheng, S., Janecek, A., Tan, Y.: Enhanced fireworks algorithm. In: Proceedings of IEEE congress on evolutionary computation. 2069–2077(2013).
21. Zheng, Y.J., Lu, X.Q., Du,Y.C.,Xue, Y., Sheng, W.G.: Water wave optimization for combinatorial optimization: Design strategies and applications. Applied Soft Computing, 83, 105611 (2019)

MISTA 2019

Research on Dynamic Balance of Mixed-model Assembly Line Based on Hybrid Genetic-Particle Swarm Optimization

Li Huang • Chunming Ye • Kun Chen • Ting Zhao • Can Qu

Abstract The dynamic balance of mixed-model assembly line is a challenging and practical value topic in the field of industrial engineering. In the process of balancing mixed-model assembly line, balance is affected by many dynamic factors, which makes the balance of initial mixed-flow assembly line no longer effective. In this paper, Mixed-model relay assembly line of M Company is taken as a research object and the hybrid Genetic-Particle swarm optimization algorithm is used to optimize and improve it. The experimental results show that, compared with standard genetic algorithm and standard particle swarm optimization, the hybrid genetic particle swarm optimization can obtain better production rhythm, smoothness index between stations and adjustment costs, and has advantages in gaining solutions effectively and efficiently.

Keywords: Mixed-model assembly line Dynamic balancing Genetic-Particle swarm optimization

Li Huang
Bussiness School, University of Shanghai for Science and Technology
School of intelligent manufacturing, Panzhihua University
E-mail: lihuang7080@163.com

Chunming Ye
Bussiness School, University of Shanghai for Science and Technology
E-mail: yechm6464@163.com

Kun Chen
Chengdu Jiepu Science and Technology Limited Company
E-mail: kun_chen0508@jabil.com

Ting Zhao
School of intelligent manufacturing, Panzhihua University
E-mail: 17308148246@139.com

Can Qu
School of intelligent manufacturing, Panzhihua University
E-mail: acan1043059763@163.com

1 Introduction

Green manufacturing with multi-varieties and small batches has become the mainstream production trend in the current market environment. Mixed-model assembly line can quickly respond to the diverse needs of customers without increasing product inventory and has been adopted by many manufacturing enterprises. Therefore, the research on mixed-model assembly line has attracted many scholars and practitioners and has also achieved rich research results.

The mixed-model assembly line balancing (MALB) problem was firstly researched by Thomopoulos [1]. Extensive literature review on MALB problems had been well done by Becker and Scholl [2] and Boysen, Fliedner and Scholl [3]. The objectives of rebalancing in mixed-model assembly line generally includes minimizing the number of stations, workload variation at each station [4], minimizing rebalancing cost such as labor cost and moving cost of machines [5] and maximizing line efficiency [6]. Many approaches have been proposed for solving the MALB problems. They are classed into two categories: exact and heuristic algorithm. Exact approaches include integer programming [7], goal programming [8] and branch and bound [9] and so on. Since the MALB problem has been proved to be NP-hard [4], heuristic algorithm such as genetic algorithm [10], simulated annealing algorithm [11] and Ant colony algorithm [12] are more practical for solving this problem.

Researchers from China have also made fruitful achievements in the field of mixed-model assembly line dynamic balancing. Dai Longzhou et al. [13] considered the impact of market demand change, product feature change and assembly technology progress on the production of mixed-model assembly line, and studied the dynamic rebalancing of mixed-model assembly line with the optimization objective of minimizing the balance cost. Yu Qingliang et al. [14] took the random mixed-model assembly line as the research object, analyzed the influencing factors of assembly line balance problem, and proposed the balance problem of the random mixed-model assembly line on the basis of considering the random factors.

In the real world, multi-objective optimization is considered more in the dynamic balancing of mixed lines. Therefore, more and more attention has been paid to multi-objective optimization and its algorithm. The two most classical multi-objective intelligent algorithms are NSGAII [15] and MOPSO [16].

From the literature survey, we can see that most of the research premises of mixed-model assembly line focus on the issue of certainty. However, due to product structure upgrading, assembly technology progress, equipment updating and other reasons, the balance of mixed-model assembly line is seriously affected. In this paper, we take M company relay mixed-model assembly line as the research object, using the multi-objective model and hybrid genetic-particle swarm intelligence algorithm to obtain a satisfactory process redistribution design. The experimental results show that, compared with the results of standard genetic algorithm and standard particle swarm optimization, the hybrid genetic particle swarm optimization can obtain better actual production rhythm, smoothness index between stations and adjustment costs, and has advantages in gaining solutions effectively and efficiently.

2 Problem Description and Modeling

2.1 Problem description

Due to the change of market demand, enterprises need to adjust production plan immediately, improve the existing product assembly process, or deleting the original assembly task, or adding new assembly tasks. The company under study (M company) has three kinds of relay products, whose output has changed from the original plan A: B: C=300:300:150 to A: B: C=500:300:100, and the total demand of these three kinds of relay products is 900 pieces. The above situation breaks the original mixed-model assembly line balance, it is not appropriate to continue the original distribution scheme, the mixed-model assembly line balance needs to be re-established. Consequently, workstations will now be reassigned.

2.2 Model Building

This paper takes the mixed-model assembly line of M company as an example for analysis. According to the company's recent orders, the minimal production set (MPS) of the mixed-model assembly line can be determined. A MPS starts with the first product entering the mixed-model assembly line and ends when the last product of the cycle leaves the production line.

When products enter the mixed-model assembly line in different order, there are different time differences and cost differences. Therefore, optimizing the production sequence can reduce the production cost, shorten the production cycle and improve the flexibility of the assembly line.

2.2.1 The basic assumptions of the model

The enterprise adjusts the mixed-model assembly line dynamically according to the production plan to realize the balance of the assembly line. The establishment of the mixed-model assembly line dynamic balance model is based on the following assumptions:
1) The original old mixed-model assembly line is basically balanced, and each operator can be competent for the assigned assembly tasks;
2) The same assembly tasks of different varieties are arranged in the same workstation;
3) The sequence of production operations of each single variety is known and can be integrated into a task sequence diagram;
4) The operation time of each production task is known, and the same production operation time of different varieties can be different;
5) The weighted time of production tasks in each workbench shall not exceed the takt time;
6) The difficulty of the task can be reflected by the length of the operation time.

2.2.2 Symbol definition

The related notations of the model are shown in Table 1.

Table 1 Notations of the model.

Notations	Definition	Requirements
N	the total number of production tasks	$\mathrm{N} > 0$
I	the number of activity element	$\mathrm{I} > 0$
D	the total demand	$\mathrm{D} = \sum_{q=1}^{Q} D_q$
Q	the total number of products that need to be produced in a production cycle	$\mathrm{Q} > 0$
D_q	the demand for the q product	$D_q > 0$
P_q	the proportion of product q demand to total demand	$t_{iq} \geq 0 P_q = \frac{D_q}{D}$
T_{iq}	the operation time of activity element i corresponding to product q	$T_k > 0$
T_k	the sum of the normal time of workstation k in one work cycle.	$T_t > T_k$
T_t	the time of a production cycle	$T_t > T_k$
U_k	the probability that the workstation k is in an operable state at any time in a work cycle,	$U_k = \frac{T_k}{T}$
PRE(i)	The set of tasks completed before task I according to the sequence of production tasks.	$\mathrm{I} = 1,2, \ldots \ldots \mathrm{N}$

2.2.3 Mixed-model assembly line construction

The dynamic balance problem of the mixed-model assembly line considered in this paper can be understood as follows: given the production ratio of different products, the working time

of each production task, and the number of workstations K, the optimal allocation scheme should be found to assign each production task to workstations and arrange operators under the constraint of the sequence of production tasks. Therefore, this paper mainly studies the influence of three factors on the balanced operation of mixed flow production line, such as market demand change, enterprise production plan change, and product upgrade. Under the constraints of the existing human resources and equipment, three mathematical models of the dynamic balance of the mixed-model assembly line were established with the optimization objective of minimizing the takt time, the smoothness index between workstations and the rebalancing cost. The model is as follows:

$$\text{MinCT} = \min \max_{1 \le k \le K} \sum_{i=1}^{N} \sum_{q=1}^{Q} P_q t_{iq} X_{ik} \tag{1}$$

$$\text{MinSI} = \sqrt{\frac{1}{K} \sum_{k=1}^{K} \left(\sum_{i=1}^{N} \sum_{q=1}^{Q} P_q t_{iq} X_{ik} - \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{q=1}^{Q} P_q t_{iq} X_{ik} \right)^2} \tag{2}$$

$$\text{MinRC} = \sum_{K=1}^{K} \sum_{l=1}^{L} \sum_{i=1}^{N} C_{ikl} Z_{kl} \tag{3}$$

$$Min\ Z \quad = a * CT + b * SI + c * RC \tag{4}$$

$$C_{ikl} = \sum_{q=1}^{Q} t_{iq}\ max\{X_{ik} - h_{il}, 0\}, \forall i, k, l \tag{5}$$

S.t.

$$\sum_{i=1}^{N} X_{ik} = 1, i = 1,2, \dots N \tag{6}$$

$$\sum_{K-1}^{K} \left(K_{Xik} - K_{Xjk} \right) \le 0, \forall i \in PRE(j) \tag{7}$$

$$\sum_{i=1}^{N} \sum_{q=1}^{Q} P_q\ t_{iq} X_{ik} \le CT_0, K = 1,2, \dots K \tag{8}$$

$$X_{ik} \in \{0,1\} \forall i, k \tag{9}$$

The objective of the model is to find the weighted minimum of production rhythm, workstation smoothness index and rebalancing cost, as shown in Eq. (4) Among them, a, b and c are the adjustment coefficients, which meet the requirement that the sum of a and b and c is 1. The minimization of each sub-objective is correspondingly shown in Eq. (1)-(3), respectively. In the new balance, if worker L is assigned to K (k = 1, 2, n) workbenches, for a single task i, the task time that the worker needs to learn, as shown in Eq. (5). The constraints of the model is seen as Eq. (6)-(9). Eq. (6) is to ensure that each production task is only assigned to one workstation. Eq. (7) indicates that the assignment task must meet the production task sequence constraint; Eq.(8) means that the total operating time of each workstation cannot exceed the

planned production beat; Eq.(9) indicates that in the new balance, when the i-th assembly task is assigned to the k-th workstation, the decision variable X_{ik} is 1, otherwise 0.

2.3 The analytic hierarchy process (AHP) calculates weights

Based on the dynamic balance problem of multi-objective mixed flow production line, the weight value corresponding to each objective needs to be calculated. Analytic hierarchy process (AHP) is a systematic comprehensive evaluation method that can solve multiple goals, levels and schemes. First, the comparison judgment matrix is constructed by means of mutual comparison of target values, then the weight of each target is taken as the component of the maximum eigenvector, and the evaluation target is evaluated comprehensively according to the weight of each index and index value. Weight calculation of AHP follows the following steps [17]:

Step I: Optimize the judgment matrix construction of the importance degree of the objective

Rebalancing costs to minimize, can bring more profits for the enterprise. Thus, rebalancing costs are more important than actual beats and the workstation smoothing index. Minimization of actual production beat can improve mixed-model assembly line production efficiency, which is a key objective to ensure timely delivery of products. Therefore, the actual production beat is more important than the workstation smoothing index. The multi-objective judgment matrix can be shown in table 2:

Table 2 Judgment matrix of multiple objectives.

Optimization Objectives	Actual production time	Workstation smoothness index	Rebalancing cost
Actual production time	1	4/3	1/3
Workstation smoothing index	3/4	1	1/4
Rebalancing cost	3	4	1

Step II: Calculation of objectives weight

According to the above multi-objective judgment matrix, the weights of the three optimization objectives can be calculated by MATLAB. It is concluded that value of the weight as follows: w= [0.2105, 0.1579，0.6316].

Step III: Consistency check

If the consistency ratio CR is less than 0.1, the judgement matrix is considered to be in line with the actual situation, otherwise, the judgement matrix needs to be reconstructed until CR is less than 0.1. The result of calculation is CR < 0.1, the consistency check is passed. Therefore, the objective weights of actual production rhythm, workstation smoothness index and rebalancing cost optimization are as follows: a = 0.2105, b = 0.1579, c = 0.6316.

3 Optimization design of particle swarm optimization and genetic algorithm

Particle swarm optimization (PSO) is simple and has few control parameters. However, particle swarm optimization (PSO) also has some disadvantages, such as slow convergence speed, falling into local optimum and so on. Considering the advantages of genetic algorithm in global search, genetic algorithm can be integrated into particle swarm optimization (PSO) to make up for the shortcomings of PSO. The combination of the two algorithms can make better use of the advantages of each algorithm and achieve the purpose of promoting the advantages and complementing the disadvantages.

A large number of relevant literatures have been reviewed. Up to now, a variety of particle swarm integration-genetic hybrid algorithms have been developed in domestic and foreign academic circles [18-21]. In this paper, the combination of the two algorithms can be classified into three categories: horizontal combination, vertical combination and cross combination.

(1) Horizontal combination: the population is divided into two categories: one is optimized by particle swarm optimization and the other is optimized by genetic algorithm. Zhu Chuntao etc. [18] in the process of the hybrid algorithm optimization is put forward, with the increasing of optimal iterative dynamic change of the population is divided into two parts, the part is the child population 1, using particle swarm optimization algorithm for optimization, and the other part is the population of 2, using genetic algorithm optimization, in order to ensure the two algorithms can always improve their own shortcomings, two child population at the same time exist in the optimization process.

(2) Vertical integration: in the maximum iterative algebra, one optimization algorithm is used in the former part of the iterative algebra, and another optimization algorithm is used in the latter part of the iterative algebra. Marlene proposed in the literature [19] in the early stage of the optimization, by using the genetic algorithm has better global searching ability, random search feature, crossover and mutation operators have features such as, using genetic algorithm to first after certain iterative algebra or to specify the precision, the multi-objective optimization problem of initial solution space, and then late optimization using particle swarm optimization algorithm convergence speed, strong local search ability characteristics, GAPSO algorithm is obtained by combination of two parts.

(3) Crossover: the idea of using another optimization algorithm in one optimization algorithm. Jia Jianfang [20] et. al proposed a hybrid genetic particle swarm optimization algorithm, in which all particles carried out crossover mutation, and compared the fitness of each particle before and after crossover mutation. If the fitness is better than before the crossover mutation, replace, otherwise, do not replace.

3.1 Hybrid genetic particle swarm optimization algorithm operation design

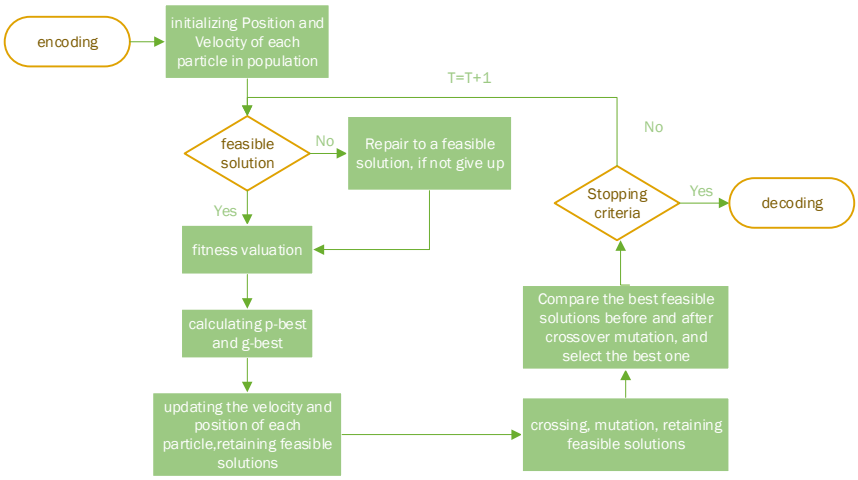


Figure 1 Flowchart of hybrid genetic-particle swarm optimization algorithm.

3.2 Encoding and fitness evaluation

(1) Integer encoding

Since the sequence number of production task is represented by integer, the solution adopts integer encoding. The total number of production tasks is taken as the number of chromosome genes. The number on each gene site indicates which task to perform. And each number appears only once on the chromosome. If there are 9 production tasks on the production line, the coding form of a randomly generated chromosome is shown in figure 2. Work tasks are allocated to each workstation in sequence according to the production order, as shown in figure 3.

①②③⑦⑧④⑤⑥⑨

Figure 2 Gene coding.

Figure 3 Production tasks assigned to workstations.

(2) Fitness evaluation

Determine the optimization objectives studied (actual production beat CT, inter-station smoothing index SI, and then adjust the cost RC), so as to find the optimal solution of the whole population. The comprehensive evaluation function is:

$$Z_{min} = a \times CT + b \times SI + c \times RC \qquad (10)$$

3.3 Crossover and mutation

(1) Two-point crossover

According to the crossover probability, the crossover operation was carried out on the two chromosomes participating in the crossover, father 1 and father 2. Randomly generated, two intersections 2, 6; the two intersections divide the parent 1 and parent 2 chromosomes into three segments called E1, E2, and E3; The gene segments that need to be crossed in father generation 1 are production task 3, 7, 8, 5 and 1. The sequence of production task 3, 7, 8, 5 and 1 in father generation 2 is searched, and the E2 gene fragment in son generation 1 is obtained, namely, 7, 1, 8, 3 and 5. Similarly, the sequence of genes in father generation 2 in father generation 1 was searched, and E2 gene fragment in child generation 2 was obtained. The crossover operation is shown in figure 4.

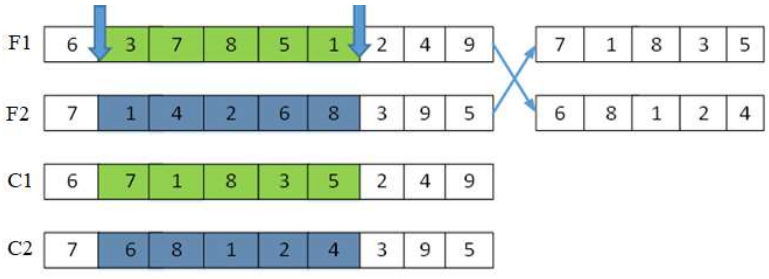


Figure 4 Two-point crossover.

(2) Two-point mutation

According to the mutation probability, the mutation operation was carried out on the parent chromosome 1 participating in the mutation. Two variation points 3,6 were randomly generated. The mutation point exchanges the gene points 7 and 1 of the parent generation 1 to obtain the gene results 1 and 7 of the child generation 1. The mutation operation is shown in figure 5.

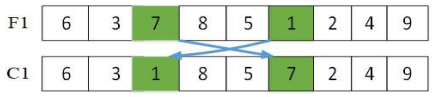


Figure 5 Two-point mutation.

4 Study on dynamic balance of mixed-model assembly line

4.1 Analysis of the current situation of the final production line

The company follows an order-by-order approach, calculating the number of products the company needs to complete each day based on the customer's order requirements. This paper investigated the product demand of customers in recent months and calculated the daily planned output of relay products, as shown in table 3:

Table 3 Daily planned production of products.

model	A	B	C
Daily planned output(pieces)	300	300	150

It can be seen from table 3 that the total demand for relays of the company is 750 pieces per day. In one production cycle, the output of relays of A, B and C is 300 pieces, 300 pieces and 150 pieces respectively, with the corresponding specific proportion of Q_a: Q_b: Q_c =2:2:1. One shift per day, the effective working time of each shift is 450 minutes, and the production preparation time is 30 minutes. Taking one day as a planned production cycle, the average production beat C of the mixed-model assembly line is:

$$C = \frac{T}{\sum_{i=1}^{n} Ni} = \frac{8\times3600-30\times60}{300+300+150} = 36\ s/pieces \qquad (11)$$

Where, T is the effective working time within a planned production cycle; Ni is the output of product I in the planning period of the production task; N is the number of varieties to be produced in the planned production cycle of the mixed-flow unit production line.

At present, there are 6 workstations and 24 production tasks on the line, and the balance efficiency is 88%, that is, the mixed flow mixed-model assembly line is relatively good, and the smoothing index is 4.68. The working time of each workstation is shown in figure 6. In order to make the working procedure of the workstation more intuitive and visual, the working procedure of each workstation is expressed as blue, red, green, purple and light blue in sequence, as shown in Figure 6.

4.1.1 Final mixed-model assembly line priority relationship

The three kinds of relay products needed in A, B, C of the order of production tasks, respectively, as shown in figure 7(A)-(C). Red denotes the start element, blue denotes the end element, and yellow denotes the intermediate work element.

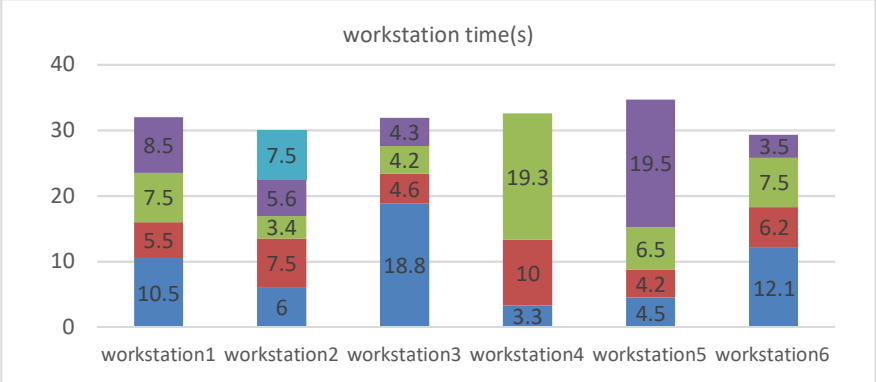
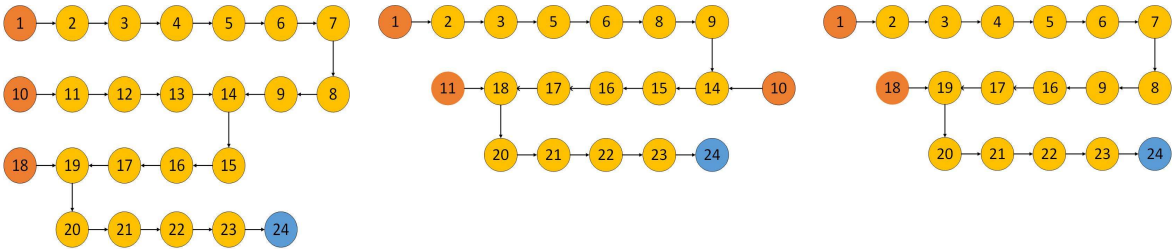


Figure 6 Time of each workstation.



(A) Product type A. (B) Product type B. (C) Product type C.

Figure 7 Production sequence of different types of products.

According to the priority assembly sequence of three different types of ABC products, the comprehensive operation sequence diagram of a product is synthesized, that is, from Figure 7 to figure 8. The above time in Figure 8 is the comprehensive operation time obtained by weighting the three products of ABC.

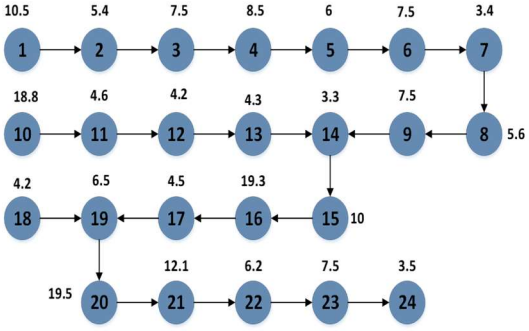


Figure 8 Mixed production sequence of different types of products.

According to the production sequence of each product shown in figure 8, the stopwatch time-measuring method is adopted to measure and calculate all operating elements of relays of three types A, B and C, and the operating time of the integrated operating elements of the production of the three relays is shown in table 4. The workstation status is shown in table 5.

Table 4 Operating time of relay production operation elements.

No.	Job element	T_A (s)	T_B (s)	T_C (s)	Integrated time $t_i = \sum_{q=1}^{Q} P_q t_{iq}$
1	coil	10.5	10.5	10.5	10.5
2	Riveting yoke, core and coil	5.4	5.4	5.4	5.4
3	Dynamic and static electric riveting	7.5	7.5	7.5	7.5
4	Spot welding	14.1	0	14.2	8.5
5	The base instrumentation	6	6	6	6
6	The electric shock	7.5	7.5	7.5	7.5
7	Hanging spring	5.6	0	5.7	3.4
8	Winding	5.6	5.6	5.6	5.6
9	Welding principle architrave	7.5	7.5	7.5	7.5
10	Circuit board welding	23.5	23.5	0	18.8
11	Clean circuit board	11.5	0	0	4.6
12	Drying circuit board	10.5	0	0	4.2
13	Visual inspection of weld toe	10.7	0	0	4.3
14	Circuit board is assembled with yoke iron and base	4.1	4.2	0	3.3
15	Weld circuit board with yoke iron and base	12.5	12.5	0	10
16	Function debugging	19.3	19.3	19.3	19.3
17	Dust removal	4.5	4.5	4.5	4.5
18	Printed label	4.2	4.2	4.2	4.2
19	With the shell	6.5	6.5	6.5	6.5
20	inspection	19.5	19.5	19.5	19.5
21	sealant	12.1	12.1	12.1	12.1
22	drying	6.2	6.2	6.2	6.2
23	packaging	7.5	7.5	7.5	7.5
24	stick labels	3.5	3.5	3.5	3.5
total		225.8	173.7	153.2	190.4

Table 5 Status table of workstation distribution.

Number of workstation	01	02	03	04	05	06
Production task	1、2、3、4	5、6、7、8、9	10、11、12、13	14、15、16	17、18、19、20	21、22、23、24
Working time /s	32	30	31.8	32.6	33.7	29.3
Number of processing workers	2	1	1	1	2	1

According to figure 7 and table 5, the time distribution of each workstation is relatively balanced at present, and the existing production the line balance is in a more efficient state.

4.2 Dynamic balance of mixed-flow line

Change due to market demand, the enterprise immediately adjust the production plan, improve the existing product production process, cut off the original production task 14, 25, 26 new production task, and the production of three kinds of relay products by the original plan A: B: C = 300:300:150 into the A: B: C = 500:300:100, three kinds of relay products total demand for 900 pieces. The above situation broke the original balance of the production line. It is not appropriate to continue the original distribution plan, and the balance of the mixed-model assembly line needs to be re-established. In order to realize the rebalance of the production line, the original mixed flow mixed-model assembly line will be optimized.

4.2.1 Research on rebalancing of mixed production line

The comprehensive work priority on the production line is shown in figure 8. Under the existing conditions, the existing 6 workstations on the line are effectively utilized to realize the balance of the mixed flow mixed-model assembly line again. The operation time of each newly added production task is shown in table 6 the operation time of three original production tasks was changed due to the introduction of new production equipment and the adoption of new operation methods. The operation time was changed as shown in table 7.

Table 6 Operating time of newly added production element.

Element i	Operating time of each element (s)		
	T_{iA}	T_{iB}	T_{iC}
25	4.1	4.1	0
26	3.3	3.4	3.3

Table 7 New operation time of production element.

Element i	Operating time of each element (s)		
	T_{iA}	T_{iB}	T_{iC}
15	10.5	10	0
18	4.7	4.7	4.7
20	20.5	20.5	20.4

In order to study the realization of the dynamic balance of the mixed flow production line, the production tasks of different types of products in the mixed flow production line were integrated into the balance of the mixed-model assembly line of a single variety. In the rebalance, the production ratio of each product on the production line was A: B: C=500:300:100, and the weight Q_a: Q_b: Q_C=5:3:1. The assignment time of task I in the figure is calculated according to$t_i = \sum_{q=1}^{Q} P_q t_{iq}$. For convenience of expression, the deleted production task 14 is still recorded in the table, but its operation time is set to zero, as seen in figure 8. Note that the element 14 in red indicates the deleted work elements, the element 25 and 26 in blue indicate the added work elements, and the green indicates the unchanged work elements. The operating time of each production task calculated is shown in table 8.

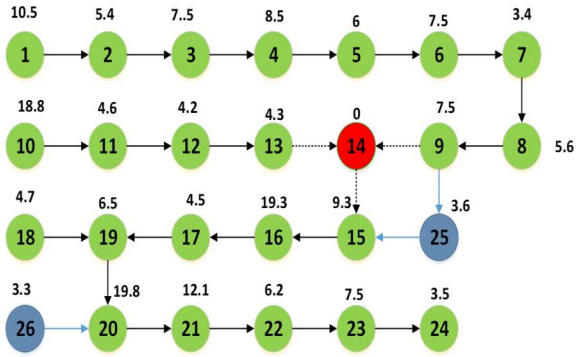


Figure 8 Changed priority sequence of integrated elements.

Table 8 Integrated operating time of each production element.

Element i	T_A (s)	T_B (s)	T_C (s)	T_{inte} (s)
1	10.5	10.5	10.5	10.5
2	5.4	5.4	5.4	5.4
3	7.5	7.5	7.5	7.5
4	14.1	0	14.2	8.5
5	6	6	6	6
6	7.5	7.5	7.5	7.5
7	5.6	0	5.7	3.4
8	5.6	5.6	5.6	5.6
9	7.5	7.5	7.5	7.5
10	23.5	23.5	0	18.8
11	11.5	0	0	4.6
12	10.5	0	0	4.2
13	10.7	0	0	4.3
14	0	0	0	0
15	13	12.5	0	9.3
16	19.3	19.3	19.3	19.3
17	4.5	4.5	4.5	4.5
18	4.7	4.7	4.7	4.7
19	6.5	6.5	6.5	6.5
20	19.8	19.8	19.8	19.8
21	12.1	12.1	12.1	12.1
22	6.2	6.2	6.2	6.2
23	7.5	7.5	7.5	7.5
24	3.5	3.5	3.5	3.5
25	4.1	4.1	0	3.6
26	3.3	3.4	3.3	3.3
Total				194.1

4.2.2 Optimization results of dynamic balance of mixed production line

In this paper, the standard genetic algorithm(GA), the standard particle swarm optimization (PSO) algorithm and the hybrid genetic-particle swarm optimization (MSO) algorithm are respectively selected to solve the above hybrid mixed-model assembly line dynamic balance problem, in order to verify that the hybrid genetic-particle swarm optimization algorithm designed in this paper can effectively solve the established dynamic balance model. The parameter Settings of the three intelligent optimization algorithms are shown in table 9. The three algorithms study the dynamic balance of the mixed flow production line, and the optimal solution optimization results are shown in table 10.

Table 9 Parameter Settings of the three algorithms.

Algorithm	Population scale	Terminated iterations	Weight values of objective			Other evolutionary parameter Settings
			a	b	c	
Standard GA	100	500	0.2105	0.1579	0.6316	pc=0.6,pm=0.05
Standard PSO	100	500				$c_1=c_2=2$,w=0.8
MSO algorithm	100	500				pc=0.6,pm=0.05, $c_1=c_2=2$,w=0.8

Table 10 Dynamic balance optimization results of mixed loading line.

Equilibrium solution	CT（s）	SI (Smooth index)	RC (Readjustment cost)	Z (Overall objective of optimization)
The ratio of products A, B and C in the initial balance scheme is 2:2:1	36	4.68	_	_
The ratio of dynamic balance scheme products A, B and C is 5:3:1 (Standard GA)	33.5	1.62	42.3	34.02
The ratio of dynamic balance scheme products A, B and C is 5:3:1 (Standard PSO)	34	1.87	42.8	34.48
The ratio of dynamic balance scheme products A, B and C is 5:3:1 (MSO algorithm)	32.7	1.29	41.1	33.04

From table 10, it can be seen the solution results of hybrid genetic-particle swarm optimization algorithm to the model are significantly better than those of standard genetic algorithm and standard particle swarm optimization algorithm. Therefore, the hybrid genetic - particle swarm optimization algorithm can obtain a more satisfactory solution. The comparison of solution efficiency of the three algorithms is shown in figure 9. The task assignment of each workstation in the optimal result obtained by the hybrid genetic particle swarm optimization algorithm is shown in figure 10.

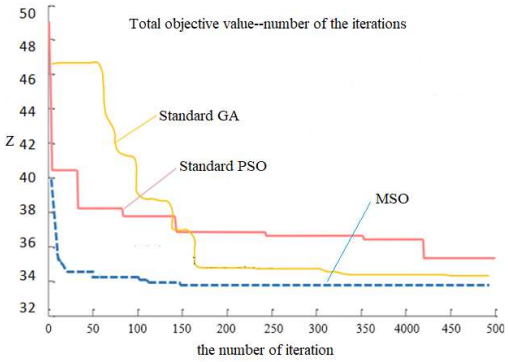


Figure 9 Comparisons of total objective value and iteration by three algorithms.

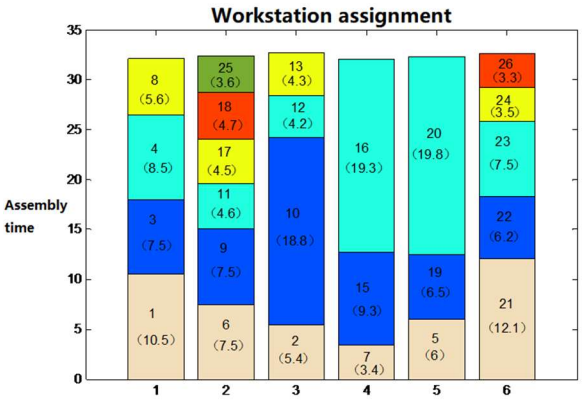


Figure 10 Task assignment on each workstation under dynamic balance.

After optimization of the hybrid genetic-particle swarm optimization algorithm, the layout of the mixed-model assembly line is rearranged as shown in figure 11.

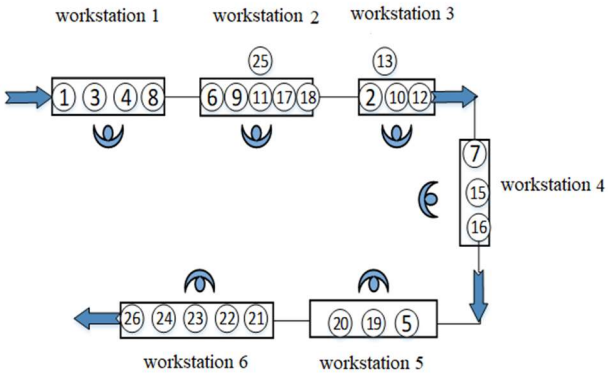


Figure 11 Optimized layout of mixed-model assembly line.

By comparing the improved layout before and after, it was found that the optimized layout has three improvements: (1) Removing the track conveyor belt to reduce the handling distance between workstation 1 and workstation 2; (2) Workstations 3, 4 and 5 adopt U-shaped layout. Operators work in a wide range and can help each other to complete production tasks; (3) Reducing the number of operators on the mixed-model assembly line by two, thus saving two labor costs.

4.2.3 Comparison and evaluation of optimization results

The optimization results of standard GA, standard PSO, literature [30], and MSO algorithm for solving the dynamic balance problem of multi-objective mixed-model assembly line, as seen in table 11. By comparing each algorithm with the original data in literature [30], it can be found that:

(1) According to the optimization results of genetic algorithm, the actual production beat was reduced from 36s to 33.5s, a decrease of 6.9%; the smoothing index decreased by 65.4% from 4.68 to 1.62. The number of operators decreased from 8 to 6, a reduction of 25%; the production efficiency of the product increased from 88.86% to 96.56%, an increase of 7.7%.

(2) Optimization results of particle swarm optimization algorithm showed that the actual production beat was reduced from 36s to 34s, a decrease of 5%; the smoothing index decreased by 60% from 4.68 to 1.87. The number of operators decreased from 8 to 6, a reduction of 25%; the product production efficiency increased from 88.86% to 95.15%, which increased by 6.29%.

(3) According to the optimization results of hybrid genetics-particle swarm optimization algorithm, the actual production beat decreased from 36s to 32.7s, a decrease of 9.1%s; the smoothing index decreased from 4.68 to 1.29, a decrease of 72.4%. The number of operators decreased from 8 to 6, a reduction of 25%; the product production efficiency increased from 88.86% to 98.93%, which increased by 10.07%.

Table 11 Optimization results of each algorithm.

	M	CT	SI	RC	Z	Product production efficiency A	Number of operators
Standard GA	6	33.5	1.62	42.3	34.02	96.56%	6
Standard PSO	6	34	1.87	42.8	34.48	95.15%	6
The literature [30]	6	36	4.68			88.86%	8
MSO algorithm	6	**32.7↓**	**1.29↓**	**41.1↓**	33.04	**98.93%↑**	6

From figure 12 to 16, we can see that MSO algorithm optimization results of each objective are better than that by the standard GA and standard PSO. This paper established the multi-objective mathematical model and proposed MSO algorithm, which can jump out of local optimal solution. Under the condition that the theoretical production rhythm is satisfied, the minimum number of workstations is obtained, the actual production rhythm is minimum, the smoothness index between workplaces is smaller, the rebalancing cost is minimum, and the production efficiency of each product in each workstation is higher.

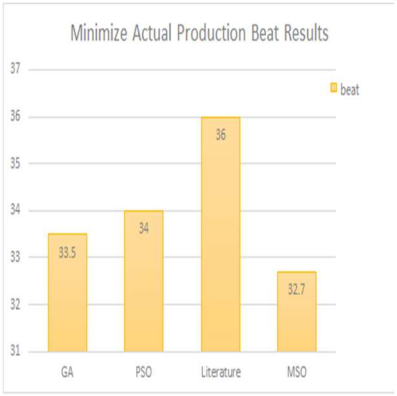


Figure 12 Comparison of minimum actual production beat.

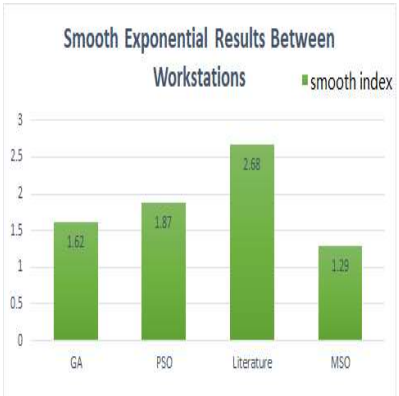


Figure 13 Comparison of smoothing index between workstations.

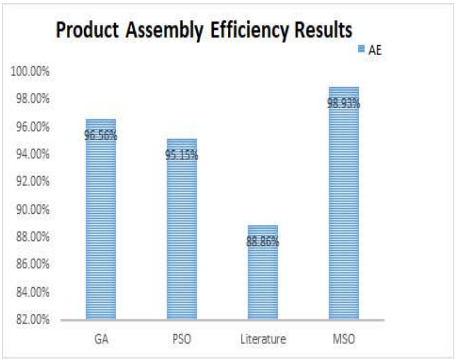


Figure 14 Comparison of product production efficiency.

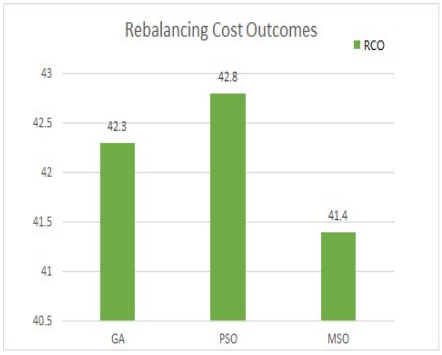


Figure 15 Comparison of rebalancing cost.

Figure 16 Comparison of general goal comparison chart.

4 Conclusion

Quick response to changing customer needs is an important guarantee for enterprises to occupy market share under fierce competition. Therefore, the mixed-model assembly line with high flexible control is widely used by many manufacturing enterprises. The mixed-model assembly line built in the early stage originates from the prediction of the current market demand, but with the passage of time, the mixed-model assembly line balance is affected by many factors, resulting in the production line balance efficiency is getting lower and lower, or even invalid. Dynamic market environment changes, product manufacturing process improvements, production technology and means of update and progress and other factors have a great impact on mixed-model assembly line balance. Therefore, it is necessary to study the dynamic equilibrium of mixed-model production line. This paper uses hybrid Genetic-Particle swarm optimization to solve the problem. The main research are summarized as follows:

(1) Making the best use of the existing human and equipment resources of enterprises, allocating production tasks, actual production time, load fluctuation between workstations and minimizing rebalancing cost of the target mixed-model assembly line, a dynamic balancing mathematical model is established.

(2) According to the standard PSO, standard GA is easy to obtain the local optimal solution, this paper proposes a hybrid MSO algorithm, a mixed strategy is the two kinds of algorithm for intersection, randomly generated initial feasible group, to speed up the convergence speed of the

particle swarm, introducing genetic algorithm crossover and mutation strategy, avoid the particle swarm into local optimum.

(3) In the dynamic balance problem of mixed-model assembly line, a multi-objective mixed-model assembly line model and a hybrid MSO algorithm are used to study the production line of M relay. Comparing standard GA, standard PSO and hybrid MSO algorithm, the experimental results show that hybrid MSO algorithm can achieve better actual production rhythm, adjust between smoothing indicators, and adjust the cost again. It has advantages in solving quality and efficiency.

(4) By optimizing the mixed production line, the actual production beat decreased from 36s to 32.7s, reducing by 9.1%. The smoothing index decreased from 4.68 to 1.29, a decrease of 72.4%. The number of operators decreased from 8 to 6, a reduction of 25%. The product production efficiency increased from 88.86% to 98.93%, which increased by 10.07%. The results show that the hybrid MSO optimization algorithm can achieve good results and has practical value.

Acknowledgements

The authors would like to thank the anomymous referees for valuable suggestions. This paper was founded by the National Natural Science Foundation of China (NO.71840003), Science and Technology Development Funding Projects of University of Shanghai for Science and Technology (2018KJFZ043), Research Funding Project of Panzhihua University (20172D005, 20172D006).

References

1. Thomopoulos, N. T. Line balancing -sequencing for mixed- model assembly. Management Science, 14(2), 59 75. (1967)
2. Becker, C., & Scholl, A., A survey on problems and methods in generalized assembly line balancing. European Journal of Operational Research, 168(3), 694-715, (2006)
3. Boysen, N., Fliedner, M., & Scholl, A., A classification of assembly line balancing problems. European Journal of Operational Research, 183(2), 674 -693, (2007)
4.Caijun Yang, Jie Gao, Linyan Sun, A muti-objective genetic algorithm for mixed-model assembly line rebalancing, Computers & Industrial Engineering, 65, 109-116, (2013)
5. Jing Zha, Jian-jun Yu, A hybrid ant colony aalgorithm for U-line balancing and rebalancing just-in-time production environment, Joural of Manufacturing Systems, 33, 93-102, (2014)
6. Rabbani M, Kazemi SM, Manavizadeh N. Mixed model U-line balancing type-1problem: a new approach. Journal of Manufacturing System, 31(2):131–8, (2012)
7. Urban TL. Note. Optimal balancing of U-shaped assembly lines. Management Science, 44(5):738–41, (1998)
8. Gokcen H, Agpak K. A goal programming approach to simple U-line balancingproblem. European Journal of Operational Research 2006, 171(2):577–85, (2006)
9. Scholl A, Klein R. ULINO: optimally balancing U-shaped JIT assembly lines.International Journal of Production Research, 37(4):721–36, (1999)
10. Hwang RK, Katayama H, Gen M. U-shaped assembly line balancing prob-lem with genetic algorithm. International Journal of Production Research, 46(16):4637–49, (2008)
11. Manavizadeh N, Hosseini N, Rabbani M, Jolai F. A simulated annealing algo-rithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-in-time approach. Computers & Industrial Engineering, 64(2):669–85, (2013)
12. Sabuncuoglu I, Erel E, Alp A. Ant colony optimization for the single model U-type assembly line balancing problem. International Journal of Production Economics, 120(2):287–300, (2009)

13. Longzhou Dai, Shaobo Li, Yongming Wu, Research on dynamic balance of mixed flow assembly line based on Improved Particle Swarm Optimization, Guizhou University, (2017)
14. Qingliang Yu, Research on the balance problem of random mixed flow assembly line based on Hybrid Particle Swarm Optimization, Southwest Jiaotong University, (2013)
15. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan ,"A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II "IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,VOL. 6, NO. 2, APRIL (2002)
16.Carlos A. Coello Coello, Gregorio Toscano Pulido, and Maximino Salazar Lechuga,"Handling Multiple Objectives With Particle Swarm Optimization,"IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 8, NO. 3, JUNE 2004.
17.Zhenhao Gao, Research on Enterprise Assembly Line Balance Method—Take Mx as an Example, North University of China, (2018)
18. Chuntao Zhu, Research on distribution network reconfiguration based on Hybrid PSO, Nanjing: Master's thesis of Nanjing University of technology, (2012)
19. Lin Ma, Particle swarm optimization genetic algorithm for security service Reengineering, Journal of Central South University of forestry science and technology, 27(5), (2007)
20. Jianfang Jia, Ruifeng Yang, Li Wang, Research on hybrid genetic particle swarm optimization algorithm, Automation instrument,34 (9):1-3, (2013)
21. Quangui Ni, Particle Swarm Genetic Hybrid Algorithm and its application in function optimization, Guangzhou: Master's thesis of South China University of technology, (2014)
22. Ruying Pang, Jihong Song, Work unit layout and production sequence of mixed flow unit production line. Modern manufacturing engineering, 18 (01), 1671-3133, (2014)

MISTA 2019

Forecasting port container throughput by "Ridge Regression - Seasonal Autoregression Integrated Moving Average - Support Vector Regression" approach

Shuojiang Xu • Hing Kai Chan

Abstract In this study, a hybrid "Ridge Regression - Seasonal Autoregression Integrated Moving Average - Support Vector Regression" model is proposed for port container throughput forecasting. First, the economic indicators of port hinterland are analyse by Ridge Regression (RR). Then the Seasonal Autoregression Integrated Moving Average (SARIMA) is used to reversely calculate the Gaussian White Noise. Then the Support Vector Regression (SVR) is used to generate forecasting model. The empirical results show that the proposed model is able to generate accurate forecasting results and suggest that ridge regression is suitable to deal with economic indicators of port hinterland with high multicollinearity.

Keywords: Container, Ridge Regression, SARIMA, Support Vector Regression, Time Series Forecasting

1 Introduction

Since the 15th century, the booming development of shipping industry has greatly changed the human society and natural landscape [1]. Compared with land transportation and air transportation, water transportation has the advantages of low cost and large transportation volume, therefore, water transportation plays an important role in international trade. Although water transportation has many advantages, but there are still some limitations, such as climate and port restrictions, cannot achieve "door to door" direct service, low speed [2, 3, 4]. In addition, the development of high-speed railway transportation has brought great challenges to traditional shipping [5].

The shipping authority of each port publishes statistical indicators every year, such as cargo at throughput port, container throughput, passenger in-and-out etc. These indicators are used to

Shuojiang Xu
International Doctoral Innovation Centre & Nottingham University Business School China, University of Nottingham Ningbo China
E-mail: Shuojiang.Xu@nottingham.edu.cn

Hing Kai Chan
Nottingham University Business School China, University of Nottingham Ningbo China
E-mail: Hingkai.Chan@nottingham.edu.cn

represent the performance of the port and its hinterland. Therefore, it is important to build accurate forecasting model for these indicators, which is able to assist the management of ports, shipping companies and freight forwarders, such as improve capacity and service quality, reduce operating costs and environmental impact. Among all the indicators, container throughput is a very important indicator, because container throughput and the industrial development of the port hinterland are directly related. Therefore, in this study, we focus on the forecasting model of port container throughput.

This study proposes a hybrid RR-SARIMA-SVR (Ridge Regression - Seasonal Autoregressive Integrated Moving Average - Support Vector Machine) to forecast the port container throughput. To the best of our knowledge, this is the first-of-its-kind that uses ridge regression to select hinterland economic data as the input variables of SVR. The steps of the model are as follows: First, the Ridge Regression (RR) is used to select final hinterland economic indicators. Second, the SARIMA is used to reversely calculate the Gaussian White Noise. Third, the SVR is used to build forecasting model for the port container throughput.

The rest of this manuscript is organised as follows. Literature review is conducted in section 2 to present the research progress of container throughput forecasting. In section 3, the related techniques are presented. Section 4 demonstrates the proposed RR-SARIMA-SVR approach in detail. Section 5 presents the results of empirical study on Ningbo-Zhoushan Port. Section 6 discusses the results in detail. In section 7, we conclude this study and point out future directions.

2 Literature review

The forecasting of port container throughput is a hot research direction. Many experts and scholars have studied the forecasting of container throughput and proposed many novel models. These models can be divided into two parts from the direction of input variables. The first part is the univariate model with the container throughput time series as the research object, and the other one is the multivariate model with the container throughput and economic data of the port hinterland as the inputs. These two different types have their own advantages and disadvantages. The univariate model is easier to obtain data, the data structure is relatively simple. Multivariate model can introduce more information, but how to deal with complex input variables becomes a problem.

For the univariate model, Xie et al. [6] conducted data characteristic analysis on the Singapore Port and the Port of Los Angeles, and employed the idea of decomposition-ensemble to analyse the decomposed components of the original time series. Similarly, Niu et al. [6] employed variational mode decomposition (VMD) on the container throughput data of Singapore Port and Shanghai Port, and used hybridizing grey wolf optimization (HGWO) to optimise the parameters of support vector regression (SVR). The results showed their model can generate more accurate results than previous models. Xu et al. [8] proposed a hybrid SARIMA-SVR model to forecast statistical indicators of aviation industry. They first used SARIMA to fit the time series of statistical indicators and reversely calculated the Gaussian White Noise. Then the SVR was used to train the selected input variables and forecast the future values of the indicators. Their results demonstrated that the Gaussian White Noise is able to improve forecasting accuracy. In this study, we employed the SARIMA-SVR proposed by Xu et al. [8] to form part of our model.

For the multivariate model, Geng et al. [9] employed multivariable adaptive regression splines (MARS) to select the candidate input variables from the economic indicators of the port's hinterland. Then the selected input variables, previous year's port throughput and second industry value, are used to train the SVR model. The results showed that the proposed approach is able to improve forecasting accuracy. However, MARS is not able to deal with the problem of multicollinearity, which is common in economic data. The study conducted by Gu and Wang [10] demonstrated that the ridge regression (RR) is able to overcome the problem of multicollinearity and find the most related variables to container throughput. Therefore, in this study, we employed ridge regression as a variable selector to find suitable input variables for the

forecasting model. To our best knowledge, this is the first time that the ridge regression is used to build hybrid port container throughput.

3 Methodology

3.1 Ridge regression

Ridge regression is a biased estimation regression method for collinear data analysis, which is an improved least squares estimation method by giving up the unbiasedness of least-square method, losing some information, and reducing accuracy. For the ill data, ridge regression can fit the data better than least square method. The cost function of ridge regression is shown in Eq.1.

$$CF = \sum_{i=1}^{m}(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}))^2 + \lambda \sum_{j=1}^{p}\beta_j^2 \quad (1)$$

where m is the number of observations, y_i is the real value of i[th] observation, β_0 is the intercept, p is the number of input variables, β_j is the coefficient of x_{ij}, x_{ij} is the j[th] input variable of the i[th] observation, λ is an adjustment parameter, $\lambda \sum_{j=1}^{p}\beta_j^2$ is the L2 regularisation of β_j.

3.2 SARIMA-SVR

SARIMA is a statistical forecasting model, which is commonly used in many fields. Previous studies presented the promising performance of SARIMA on demand forecasting. The assumption of SARIMA is that "an observable time series in which successive values are highly dependent can frequently be regarded as generated from a series of independent shocks" [11]. The expression of SARIMA is shown Eq.2.

$$\phi_p(B) \cdot \Phi_P(B^m) \cdot \nabla^d \cdot \nabla_m^D \cdot Y_t = \theta_q(B) \cdot \Theta_Q(B^m) \cdot a_t \; CF = \sum_{i=1}^{m}(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}))^2 + \lambda \sum_{j=1}^{p}\beta_j^2 \quad (2)$$

where $\phi_p(B)$ is the autoregression part (AR), ∇^d is the integration part (I), $\theta_q(B)$ is the moving average part (MA), $\Phi_P(B^m)$ is the seasonal autoregression part (SAR), ∇_m^D is the seasonal integration part (SI), $\Theta_Q(B^m)$ is the seasonal moving average part (SMA), Y_t is the original time series, and a_t is the Gaussian White Noise.

From the Eq.2, it can be seen that the Gaussian White Noise can be reversely calculated, if the parameters of other parts are already known. Xu et al. [8] proposed a method reversely calculated the Gaussian White Noise by fitted SARIMA model.

Support vector regression a famous machine learning method, which has been widely investigated and used in different fields. It is able to capture nonlinear relationship between input vector and target vector. Another advantage of SVR over artificial neural network is that it can perform well when there are limited input variables. Therefore, in this study, SVR is used to build forecasting model for port container throughput.

4 Design of experiments

In this study, a hybrid RR-SARIMA-SVR model is proposed for port container throughput forecasting. The procedures are shown in Fig.1. and the step-by-step introduction is given below.

Step 1: The historical container throughput and economic indicators of the port city are collected.

Step 2: SARIMA model is trained by the training set to find the optimal parameters. In this study, the "auto.arima" function in "forecast" package of "R" language is used to implement

SARIMA model. The "auto.arima" function will find and return the optimal parameters automatically.

Step 3: The equation of SARIMA is defined according to the parameter provided by the second step and the a_t component is reversely calculated based on the SARIMA equation. For detailed information about how to reversely calculate the Gaussian White Noise, please refer to Xu et al.'s study [8].

Step 4: The min-max normalisation is employed to process the original data. This step will change the dimensional expression to a dimensionless expression, which is convenient to compare and weight the indexes of different units or orders of magnitude.

Step 5: The ridge regression is employed to select suitable economic indicators as the input variables for the forecasting model.

Step 6: The selected input variables are used to train SVR model and forecast future container throughput. In this study, the "e1071" package in "R" is used to build the SVR model. A grid search method is necessary to find the optimal parameters of the model. The searching ranges are $[2^{-10}, 2^{-9}, \ldots, 2^{9}, 2^{10}]$ for epsilon, cost and gamma of the SVR model. For the kernel function of the model, we select the "radial basis". A 10-fold cross validation technique is used to avoid overfitting problem.

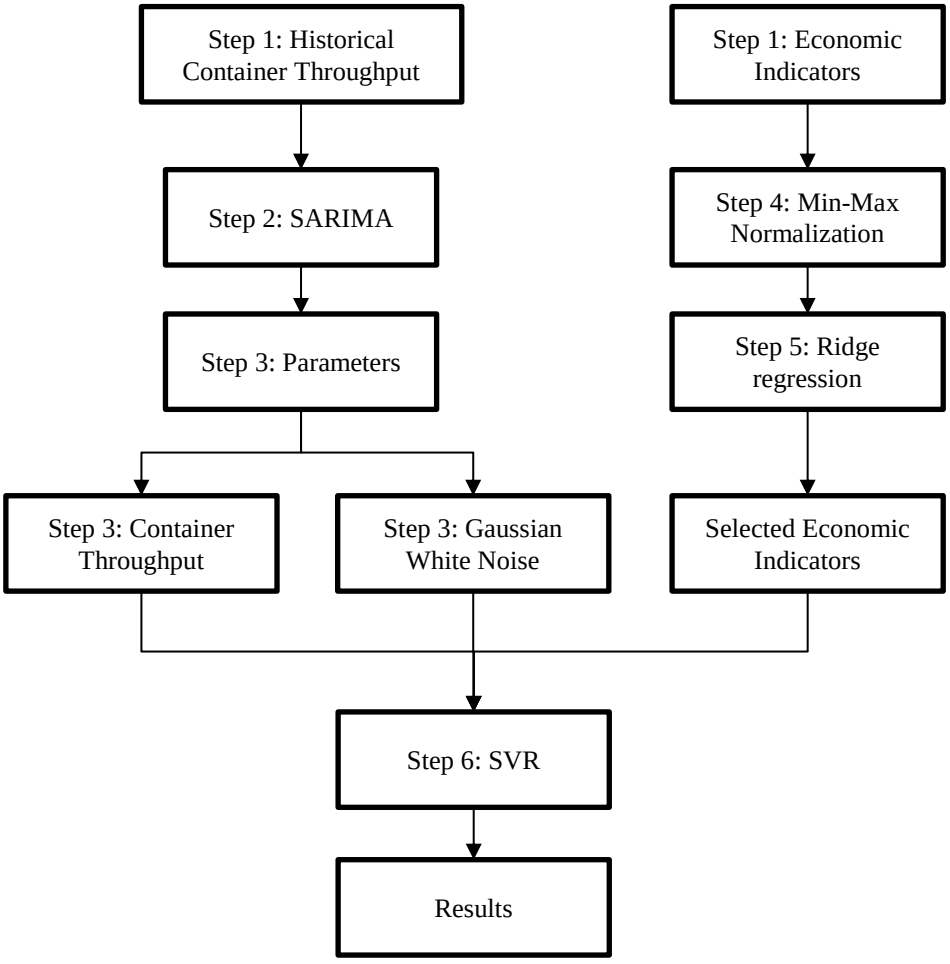


Figure 1: Procedure of RR-SARIMA-SVR approach for port container throughput forecasting

Finally, the well-trained model is applied on the testing test to see the performance of the proposed model.

5 Empirical results and discussion

5.1 Data description

China is the world's largest container handling country, with ten of the world's top 20 container ports in China [12]. China is also the world's largest manufacturing country and consumer market, which bring a large number of goods import and export business. As the world's second largest port with the total throughput, Ningbo port has a very fast growth rate. Meanwhile, it also has massive container throughput, and is also an important gateway connecting China's inland and international markets. Therefore, in this study, we collected historical container throughput data of Ningbo port and economic data of port economic hinterland. We take monthly historical container throughput from January 2009 to December 2017, with a total of 108 observations, and yearly economic indicators from 2009 to 2017, with a total of 19 observations. The first 84 observations of historical container throughput and 17 observations of economic indicators are selected as training set. The rest 24 observations of historical container throughput and 2 observations of economic indicators are selected as testing set.

5.2 Results and discussions

Before the first step in Fig.1, we conducted Cohen's Kappa test [13] to exam the level of multicollinearity. The result value is 9.59952e+17, which clearly shows the collinearity of the data. According to the Fig.1, the first step is do Min-Max normalization and ridge regression. The second step is to reversely calculate the Gaussian White Noise using SARIMA. Then the SVR is used to build forecasting model. In order to test the forecasting accuracy of the proposed model (RR-SARIMA-SVR), three other methods are used for comparison, which are ridge regression (RR), SARIMA, SARIMA_SVR. The forecasting accuracy of each method is shown in Table 1, and the time series of testing set and forecasted results are shown in Fig.2. To measure the forecasting performance of these models, several criteria are used, which are mean error (ME), root mean square error (RMSE), mean absolute error (MAE), mean percentage error (MPE), mean absolute percentage error (MAPE), autocorrelation of errors at lag 1 (ACF 1).

From the Table 1 and Fig.2, it can be seen that the proposed approach is able to generate more accurate forecasting results. For example, the MAPE of RR-SARIMA-SVR is 5.38, the others are 6.01 for RR, 7.50 for SARIMA, 5.88 for SARIMA_SVR. This result indicates that the ridge regression is able to select meaningful economic indicators to help improve forecasting accuracy. Comparing the MAPEs of RR and SARIMA, we can find that the RR alone is still able to generate relative accurate results, which also shows the ridge regression is suitable to handle the data with strong multicollinearity.

Table 1: Forecast evaluation of different models

Model	ME	RMSE	MAE	MPE	MAPE	ACF1
RR	-1.57	13.40	11.36	-1.39	6.01	0.06
SARIMA	12.66	17.70	15.11	6.08	7.50	0.61
SARIMA_SVR	8.54	13.85	11.82	3.96	5.88	0.41
RR_SARIMA_SVR	7.61	12.86	10.74	3.52	5.38	0.19

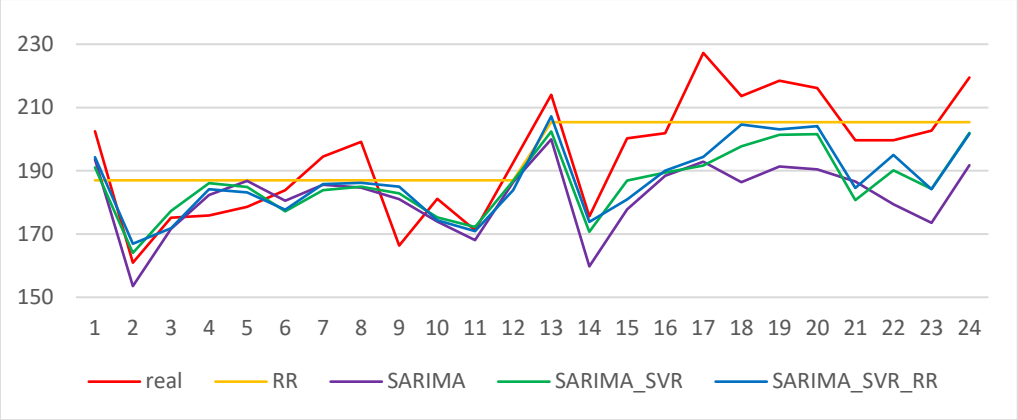


Figure 2: The time series of testing set and forecasted results

6 Conclusion

In this study, we proposed a hybrid RR-SARIMA-SVR model for port container throughput forecasting. The main contribution of this study is that we proposed a variable select mechanism using ridge regression, which is able to select meaningful variables from data with strong multicollinearity. Another contribution is that we further improved the forecasting accuracy of port container throughput.

Acknowledgements

The authors acknowledge the financial support from the International Doctoral Innovation Centre, Ningbo Education Bureau, Ningbo Science and Technology Bureau, and the University of Nottingham. This work was also supported by the UK Engineering and Physical Sciences Research Council [grant number EP/L015463/1]. This research is partially supported by Major Collaboration Programme 2017 funded by Ningbo Science and Technology Bureau (reference number: 2017D10032).

References

1. Fayle, C. E., A short history of the world's shipping industry. Routledge, London (2013).
2. Brooks, M. R., Sea change in liner shipping: regulation and managerial decision-making in a global industry. Emerald Group Publishing Limited, Bingley (2000).
3. Selkou, E., Roe, M., Globalisation, policy and shipping. Edward Elgar, Cheltenham (2004).
4. Davis, R., The rise of the English shipping industry in the seventeenth and eighteenth centuries. Oxford University Press, Oxford (2012).
5. Park, Y., Ha, H. K., Analysis of the impact of high-speed railroad service on air transport demand, Transportation Research Part E: Logistics and Transportation Review, 42(2), 95-104 (2006).
6. Xie, G., Zhang, N., Wang, S., Data characteristic analysis and model selection for container throughput forecasting within a decomposition-ensemble methodology, Transportation Research Part E: Logistics and Transportation Review, 108, 160-178 (2017).

7. Niu, M., Hu, Y., Sun, S., Liu, Y., A novel hybrid decomposition-ensemble model based on VMD and HGWO for container throughput forecasting. Applied Mathematical Modelling, 57, 163-178 (2018).
8. Xu, S., Chan, H. K., Zhang, T., Forecasting the demand of the aviation industry using hybrid time series SARIMA-SVR approach. Transportation Research Part E: Logistics and Transportation Review, 122, 169-180 (2019).
9. Geng, J., Li, M. W., Dong, Z. H., Liao, Y. S., Port throughput forecasting by MARS-RSVR with chaotic simulated annealing particle swarm optimization algorithm. Neurocomputing, 147, 239-250 (2015).
10. Gu, H., Wang, Y., Empirical analysis of the impact of industrial structure on container throughput - taking Ningbo Zhoushan Port as an example. Operation and Management, 1, 127-130 (2013).
11. Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M., Unconditional likelihood, sum-of-squares function, and least-squares. In: Balding, D.J. (Ed.), Time Series Analysis: Forecasting and Control, pp. 213–215, John Wiley & Sons, New Jersey (2015).
12. Top 50 world container ports. 2016. World Shipping Council. Retrieved from: http://www.worldshipping.org/about-the-industry/global-trade/top-50-world-container-ports
13. Wang, J., & Xia, B. (2019, August). Relationships of Cohen's Kappa, Sensitivity, and Specificity for Unbiased Annotations. In Proceedings of the 2019 4th International Conference on Biomedical Signal and Image Processing (ICBIP 2019) (pp. 98-101). ACM.

MISTA 2019

Integrated order batching and vehicle routing in cigarette distribution

Qiang Yang • Shuxian Tang •Zhenyuan Liu

Abstract: Logistics cost control is an important issue faced by tobacco industry enterprises. This paper proposes a solution framework for the order batching and vehicle routing problem in cigarette distribution. The solution framework solves the problem by designing an improved saving algorithm and matching-and-splitting rules. The results show that the proposed algorithm can save 13% of the cost and improve the space utilization of the vehicle. At the same time, this paper also determines the core factors affecting freight rate through sensitivity analysis, and proposes alternatives and recommendations for joint optimization of standard weight and freight rate through game analysis.
Key words: cigarette distribution; vehicle routing; parameter sensitivity analysis; joint optimization

1 Introduction

Logistics has become a new profit growth point for the tobacco industry. It is important to find a scientific method suitable for cigarette distribution. At present, the distribution of cigarette industry mostly uses outsourcing carriers for logistics distribution. The Tobacco Corporation has delineated a number of service packages according to the shipping warehouse and sales area. One service package contains one or more bidding sections. The carrier bidding is carried out for each bidding section, and finally the transportation area in the sales area of the bidding section is transported. The task is assigned to the carrier that won the bid.

First Author: Qiang Yang
Affiliation: School of Artificial Intelligence and Automation
Huazhong University of Science and Technology
E-mail: 1056600872@qq.com

Second Author: Shuxian Tang
Affiliation: School of Artificial Intelligence and Automation
Huazhong University of Science and Technology
E-mail: 2293614932@qq.com

Third Author: Zhenyuan Liu
Affiliation: School of Artificial Intelligence and Automation
Huazhong University of Science and Technology
E-mail: zyliu@mail.hust.edu.cn

Under the requirements of the corresponding service contract for each tender, the transportation unit price of carrier in the corresponding bidding section varies according to the transportation area, and there are single, two, and three-point transportation modes in the transportation and distribution process, when the vehicle's transportation volume does not reach the standard weight, the company needs to pay an additional subsidy fee.

In the above-mentioned logistics industry model, tobacco industry enterprises need to assign logistics center orders to specific vehicles of each carrier. The problem is essentially an order batching and vehicle allocation problem. The main thing is how to determine the appropriate order delivery plan in a scientific and reasonable way, which includes both the path planning problem (that is, determining which points the vehicle should deliver, and the path to be transported) and the problem of vehicle allocation (i.e., using that kind of Which orders are shipped by the vehicle). There are many overall design factors for the above problems, especially the large number of customers and the scattered orders. Madankumar and Rajendran[1] considered a special case of vehicle routing problem that addresses the routing problem in a semiconductor supply chain, proposed a Mixed Integer Linear Programming (MILP) models which gives better lower bounds than that by the existing work. Mancini [2] presented a Mixed Integer Linear Programming formulation and proposed a Large Neighborhood Search based Matheuristic . Coelho [3]designed a trajectory search heuristic called GILS-VND that combines Iterated Local Search (ILS), Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent (VND) procedures. Daniel et al [4] studies integrated order picking and vehicle routing problem， and designed an iterated local search algorithm for the simultaneous solution of the subproblems. Eskandarpouret al [5] developed an enhanced multi-directional local search to approximate the Pareto set.

2 Problem definition and model

The proposed model in this paper is inspired by a real life cigarette delivery company in China. The management is faced with challenges on a daily basis, regarding their routing practice. At present, there are three main aspects of the routing which cause inefficiencies for the company. We incorporate those into our problem and report possible savings and practical implications.

1) For transportation safety and customer satisfaction, each vehicle is allowed to transport orders to up to three demand points, and the tariff settlement has a subsidy rule;

2) The order quantity distribution is scattered, there are a small number of orders, and there are orders larger than the vehicle capacity (such orders need to be split);

3) The number of enterprise delivery points and customers (commercial companies) is large.

There are multiple warehouses s $\in S$ and multiple commercial companies (i.e. customers) $a \in A$ in this model, where A is a set of commercial companies and S is a set of warehouses. The commercial company issues an order o with priority ε_o, the order is from the shipping warehouse $i \in S$ to the commercial company $j \in A$ (if the shipping point of the order o is the shipping warehouse i, h_{oi} is 1; if the destination of the order o is the commercial company j, w_{oj} is 1), the order quantity is q_o, the required arrival time is tc_o, the transportation cost is $c_{ij}, i \in S, j \in A$. The carrier has a variety of models $c \in C$. The number of vehicles in type c is $\bar{V}_c$, and its capacity is Q_c, where C is the vehicle type set $C = \{c|c = 1,2, \dots, \bar{C}\}$. After the order arrives, the dispatch center will combine the orders into order group g, and then combine the orders into transport batch b. The transport batch b can be loaded at the shipping warehouse i at the time of ts_{bi}, and the loading duration is tl. It costs td_{bi} time to complete the loading and leaving node i, which is transported from the section i $(i \in S \cup A)$ to the node $j, j \in A$ is used as $te_{ij}, i \neq j$, and the arrival time is tr_{bj}. After the arrival of the destination, the unloading is completed after the length of the tu, and the goods are delivered in tr_o. At the time of cost settlement, depending on the order type, the tonnage/box conversion ratio γ_o of the order o is different, and if the vehicle load does not reach the standard weight, the calculation is based on

the corresponding standard weight. The transportation cost of the order group shipped to the commercial company j in the transport batch b is recorded as $cost_{bj}$.

Five different decision variables are used in the model: (i) binary decision variable x_{obg}, which takes the value 1 if order o is assigned to order group g of transport batch b, and 0 otherwise; (ii) binary decision variable y_{bc}, which takes the value 1 if batch b is transported by type c vehicle; (iii) binary decision variable z_{bi}, which takes the value 1 if batch b contains the order at node i, and 0 otherwise; (iv) binary decision variable z_{bgj} which takes the value 1 if the destination of order group g contained in batch b is node j, and 0 otherwise; and (v) binary decision variable u_{bij} which takes the value 1 if batch b is transported from node i to node j, and 0 otherwise;

$$\min\ cost = \sum_{b\epsilon B}\sum_{j\epsilon A} cost_{bj} \tag{1}$$

$$cost_{bj} = \begin{cases} p_{bj}\cdot\sum_{i\in S} z_{bi}d_{bij}c_{ij}\ , p_{bj} > p_{n_b} \\ p_{n_b}\cdot\sum_{i\in S} z_{bi}d_{bij}c_{ij}\ , p_{bj} \le p_{n_b} \end{cases},\ \forall b\epsilon B, j\epsilon A \tag{2}$$

$$p_{bj} = \sum_{g\in G}\sum_{o\in O} x_{obg}\cdot w_{oj}\cdot\gamma_o\cdot q_o,\ \forall b\epsilon B, j\in A \tag{3}$$

$$n_b = \sum_{j\in A} z_{bj},\ \forall b\epsilon B \tag{4}$$

$$\max\ \frac{\bar{O}-\sum_{o\epsilon O}\varepsilon_o\cdot k_o}{\bar{O}} \tag{5}$$

$$k_o = \begin{cases} 1, tr_o \le tc_o \\ 0, tr_o > tc_o \end{cases},\ o\epsilon O \tag{6}$$

Subject to:

$$\sum_{b\in B}\sum_{g\in G} x_{obg} = 1,\ \forall o\in O \tag{7}$$

$$\sum_{i\in S} z_{bi} \le 1,\ \forall b\in B \tag{8}$$

$$z_{bi} = \begin{cases} 1, \sum_{g\in G}\sum_{o\in O} h_{oi}\, x_{obg} > 0 \\ 0, \text{otherwise} \end{cases},\ \forall b\in B, i\in S \tag{9}$$

$$\sum_{j\in A} z_{bgj} \le 1, \forall b\in B,\ \forall g\in G \tag{10}$$

$$z_{bgj} = \begin{cases} 1, \sum_{o\in O} w_{oj}x_{obg} > 0 \\ 0, \text{otherwise} \end{cases},\ \forall b\in B, g\in G, j\in A \tag{11}$$

$$\sum_{c\in C} y_{bc} = 1,\ \forall b\in B \tag{12}$$

$$\sum_{b\in B} y_{bc} \le \bar{V}_c,\ \forall c\in C \tag{13}$$

$$\sum_{g\in G}\sum_{o\in O} q_o\, x_{obg} \le \sum_{c\in C} Q_c y_{bc},\ \forall b\in B \tag{14}$$

$$\sum_{i\in S\cup A} u_{bij} = \sum_{i\in A} u_{bji},\ \forall j\in A, b\in B \tag{15}$$

$$ts_{bi} + tl \le td_{bi} + M(1-z_{bi}),\ \forall b\in B, i\in S \tag{16}$$

$$td_{bi} + te_{ij} \le tr_{bj} + M\left(1-u_{bij}\right),\ \forall b\in B, i\in S\cup A, j\in A, i\ne j \tag{17}$$

$$tr_{bj} + tu \le td_{bj} + M\left(1-z_{bj}\right),\ \forall b\in B, j\in A \tag{18}$$

$$z_{bj} = \begin{cases} 1, \sum_{g\in G} z_{bgj} > 0 \\ 0, \text{otherwise} \end{cases},\ \forall b\in B, j\in A \tag{19}$$

$$tr_o - tc_o \le tm,\ \forall o\in O \tag{20}$$

$$tr_o = \sum_{b\in B}\sum_{g\in G} tr_{bj}x_{obg}w_{oj},\ \forall j\in A, o\in O \tag{21}$$

$$x_{obg}, y_{bc}, z_{bi}, z_{bgj}, u_{bij}, k_o \in \{0,1\},\ \forall o\in O, b\in B, g\in G, c\in C, i\in S\cup A, j\in A \tag{22}$$

$$td_{bi}, tr_{bj}, tr_o, cost_{bj}, p_{bj} \ge 0,\ \forall b\in B, i\in S\cup A, j\in A, o\in O \tag{23}$$

$$n_b \in \{1,2,3\},\ \forall b\in B \tag{24}$$

The Objective Function (1) aims to minimize the total cost of travel. Constraints (2)-(4) Indicates the transportation cost of the order group shipped to the commercial company j in the transport batch b. The Objective Function (5) aims to maximize weighted order satisfaction rate. Constraints (7)- ensures an order is only assigned to a group. Constraints (8)-(9) ensures that the orders in the same batch have the same starting warehouse. Constraints (10)-(11) ensures that the orders in the same group have the same destination. Constraints (12) ensures one batch corresponds to one vehicle type. Constraints (13) indicates the number of vehicles. Constraints (14) guarantee that the vehicle is not overloaded. Constraints. (15) guarantee that the number of shipments arriving and leaving the commercial company is equal. Constraints. (16)-(19) ensures that the transportation process is reasonable in time. Constraints. (20)-(21) ensure that the delay

time of the delayed order does not exceed the specified maximum delay time. Constraints. (22)-(24) describe the scope of the variable.

3 Solution methodology

This paper designs a heuristic vehicle allocation and routing planning algorithm (HVAR). The main idea behind HVAR is to reduce or even eliminate freight subsidies caused by different order sizes through order combination, and to improve vehicle utilization through the vehicle allocation mechanism. Details of HVAR are formally described in Algorithm 1

The algorithm mainly uses the key influencing factors of the distribution cost in the logistics distribution stage as the order allocation rules, and makes the order combination for each distribution starting point and forms the order transportation batch plan to determine the transportation batch of all the distribution starting points. After the plan, all vehicles are assigned to the transport batches. Finally, some batches that cannot be loaded by a single vehicle are split and assigned vehicles.

Algorithm 1 HVAR

Input： *order set* ***O****, vehicle set* ***C****, business company set* ***A****, Shipping warehouse set* ***S****, distancematrix* ***D****, single point stand weight* $\boldsymbol{t_1}$*, double point stand weight* $\boldsymbol{t_2}$*, three point stand weight* $\boldsymbol{t_3}$*, average waiting time* ***t****, average speed of trucks* ***v***

Output： *Result R*

$G, B, \mathrm{B}_{less}, \mathrm{B}_{over} \leftarrow null$
$i \leftarrow 0$
foreach $o \in O$ // Put the order in the same origin and destination at one group
  foreach $g \in G$ **do**
    if (the shipping warehouse of order *o* = the shipping warehouse of group g **and** the destination of order *o* = the destination of group g) **then**
      $g^i \leftarrow g^i \cup \{o\}$
    end
    else
      $i \leftarrow i + 1$
      $g^i \leftarrow \{o\}$
      $G \leftarrow G \cup \{g^{i+1}\}$
    end
  end
end
$B \leftarrow batch\ generation(G, D, t_1, t_2, t_3, t, v)$
foreach $b \in B$ **do**
  if (the weight of $b \leq$max{C}) **then**
    $B_{less} \leftarrow B_{less} \cup b$
  end
  else
    $B_{over} \leftarrow B_{over} \cup b$
  end
end
$R, C_{left} \leftarrow vehicleallocation(B_{less}, C)$
if$(B_{over} \neq null)$ **then**
  $R \leftarrow R \cup adjust(B_{over}, C_{left})$
end
Return *R*

3.1 Batch generation

In practice, the cost subsidy due to the quality of the order group is less than the standard weight is urgently needed to be optimized, so combine those groups into batches to reduce transportation costs. In the process of merging, first merge the two groups to find out the combination method with the largest saving value, and then judge whether there is still subsidy in the current portfolio. If it exists, continue to combine until all the groups cannot continue to combine. It is worth noting that each combination needs to do *due-time-chec*k for each order, ensure that every order can arrive in time, and a batch can accommodate up to three groups. Details of batch generation are formally described in Algorithm 2

Algorithm 2 $batch\ generation$

Input: $G, D, t_1, t_2, t_3, t, v$
Output: B

fasibel←*false*
$i, saving, saving_{temp} \leftarrow 0$
$mark1, mark2, mark3 \leftarrow null$
While $fasible \neq true$ **OR** $G = null$ **do**
  foreach $g_1 \in G$ **do**
    foreach $g_2 \in G$ **do**
      if(*due-time-check*(g_1, g_2) = *true***) then**
        $saving_{temp}$ = *cost*(g_1) + *cost*(g_2) -*cost*(g_1, g_2)
        if($saving_{temp} > saving$**) then**
          $mark1 \leftarrow g_1$, $mark2 \leftarrow g_2$
        end
      end
    end
  end
  if(*saving >0*) **then**
    $G \leftarrow G/\{g_1, g_2\}$
    $b^i \leftarrow \{g_1, g_2\}$
    if(the weight of $g_1 < t_2$ OR the weight of $g_2 < t_2$) then
      foreach $g_3 \in G$ **do**
        if(*due-time-check*(g_1, g_2, g_3) = *true***) then**
          $saving_{temp}$ = *cost*(g_1) + *cost*(g_2) + *cost*(g_3)-*cost*(g_1, g_2, g_3)
          if($saving_{temp} > saving$**) then**
            $mark3 \leftarrow g_3$
          end
        end
      end
      if(*saving >0*) **then**
        $G \leftarrow G/\{g_3\}$
        $b^i \leftarrow b^i \cup \{g_3\}$
      End
    End
    $B \leftarrow B \cup b^i$
    $i \leftarrow i + 1$
    $fasible \leftarrow false$
  end
  else
    $fasible \leftarrow true$
  end
end
return B

3.2 Vehicle allocation

All order transport batches are arranged in ascending order of batch traffic, and the vehicles are also arranged in ascending order of vehicle capacity, and the vehicles are assigned to each batch according to the order of the order transport batch. Such an allocation process can avoid a situation in which the vehicle loading rate is small during the random allocation process. The specific process is as shown in Algorithm 3。

Algorithm 3 $vehicleallocation$

Input: B_{less} ,C
Output: R ,C_{left}

$C_{left}, R \leftarrow null$
$C \leftarrow$*Sorting vehicles set in ascending order by* Q_c
$B_{less} \leftarrow$*Sorting batch set in ascending order by* p_b
foreach $b \in B_{less}$ **do**
    $r \leftarrow$*allocate the batch to the vehicle c with the closest batch capacity*
    $R \leftarrow R \cup \{r\}$
    $\bar{V}_c \leftarrow \bar{V}_c - 1$
    if $(\bar{V}_c = 0)$ **then**
        update C
    end
end
$C_{left} \leftarrow C$
return R, C_{left}

3.3 Logistics distribution plan adjustment

The quantity of the batch greatly exceeds the transportation capacity of the largest transport vehicle (about 400 boxes), so it is necessary to consider how to split these orders for transportation. Since the vehicles allocated according to the vehicle type distribution rules have leftovers, the transportation vehicles with large capacity are generally left. Therefore, the greedy search matching method is adopted, and the traffic with the larger volume is preferentially allocated among the orders that need to be split. Vehicles are allocated from the maximum number of vehicles that can be used, and are assigned to each vehicle in turn. During the split process, the order for each split must be not less than single point stand weight(12t), so there will be no freight subsidies. The specific process is as shown in Algorithm 4。

Algorithm 4 $adjust$

Input: B_{over}, C_{left}
Output: R

$R`, R \leftarrow null$
$C_{left} \leftarrow$*Sorting vehicles set in descending order by* Q_c
$B_{over} \leftarrow$*Sorting batch set in descending order by* p_b
foreach $b \in B_{over}$ **do**
    $R` \leftarrow split\ batch\ based\ on\ C_{left}\ and\ \bar{V}_c$
    $R \leftarrow R \cup R`$
    update $\bar{V}_c$
    update C
end
return R

4 Numerical experiments

4.1 Computation results

The basic parameters for solution methods are defined: standard weight for single point transport, two-point transport and three-point transport is respectively 12 tons, 10 tons and 5 tons; waiting time for unloading is 24 hours; truck velocity is 60 km/h.

Algorithm HVAR is compared with two methods: direct transport algorithm (DTA) and history scheduling scheme (HSS). In the DTA, each order group applies single point transport. HSS means scheduling by experience. Note that there are two modes of HVAR, one considers the current transport mode and the other is in an intensive transport mode. Fig. 1 shows transportation costs in 12 months solved by DTA, HVAR and HSS. As shown in Fig. 1, the transportation cost of HVAR is much lower than DTA and HSS.

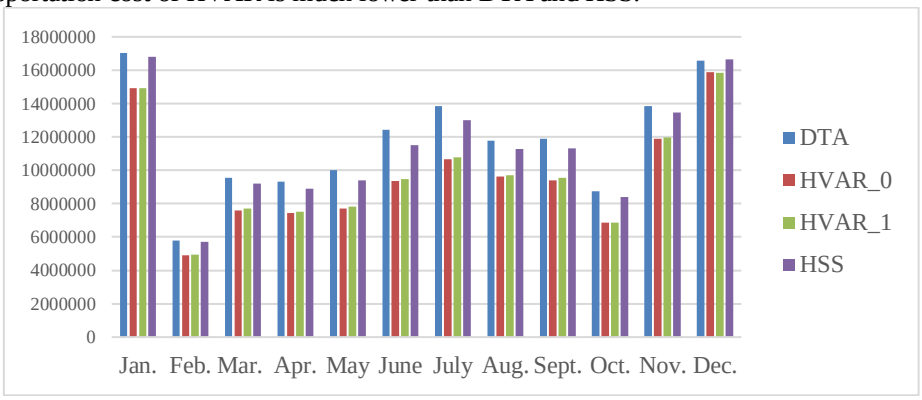
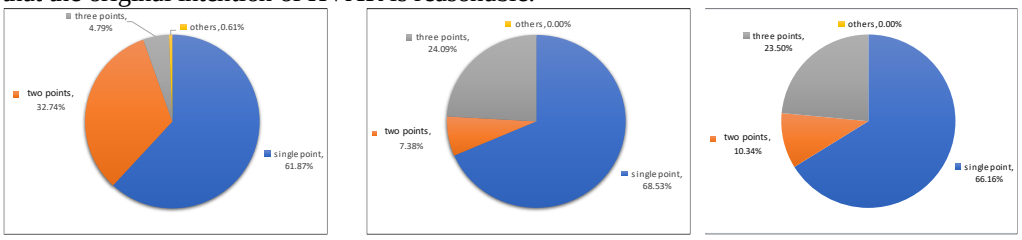


Fig. 1 Transportation costs solved by three methods

Compared with DTA, HSS saves 3.67% of the cost, which is about 5.16 million yuan. However, the cost of HSS is probably higher than DTA due to the strong human factors. For example, the cost of HSS in December wasted 90,000 yuan. The performance of HVAR is the best. In details, HVAR saves 16.91% of the cost compared with DTA and saves 13.75% of the cost in contrast to HSS, which is about 18.65 million yuan.

Fig. 2 illustrates that HVAR greatly increases the proportion of three-point transport and appropriately reduces the number of single point transport and two-point transport, which proves that the original intention of HVAR is reasonable.



(a)HSS (b)HVAR(current transport model) (c) HVAR(intensive transport model)

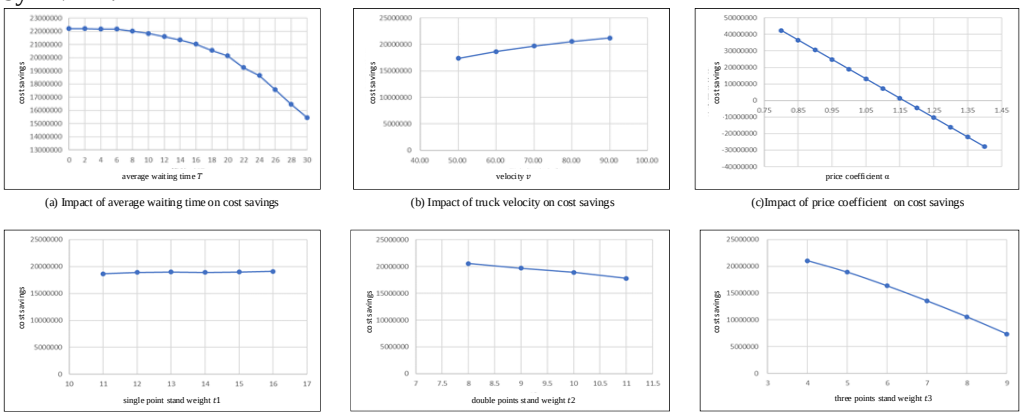
Fig. 2 Analysis results of transport structure in three methods

4.2 Parameter sensitivity analysis

This system has six important parameters: standard weight of single point transport t_1, standard weight of two-point transport t_2, standard weight of three-point transport t_3, average waiting time T, average truck velocity v and freight rate α. For each transport mode, if the transport weight of a truck is lower than the standard weight, the calculation of transport cost is on the basis of the standard weight. Otherwise, the transport cost calculates according to the actual weight. Apart from transportation time, there are waiting time for unloading and other

time in the multi-point transport, which are replaced by the average waiting time T. The time arriving at the next city = distance/average velocity + average waiting time, which is mainly used to ensure the time safety between multi-point transport. Average truck velocity v is related to the feasibility of incorporating orders. Freight rate α reflects the transport price fluctuations, which directly affects the transport price.

Keeping other parameters constant, take the values of T in 0 to 30 hours, v in 50, 60, 70, 80, 90 (velocity unit: km/h), t_1 in 11 to 16 tons, t_2 in 8 to 11 tons, t_3 in 4 to 9 tons and α in 0.8 to 1.4. Fig. 3 shows the cost saving controlled by different system parameters. The cost is solved by HVAR.



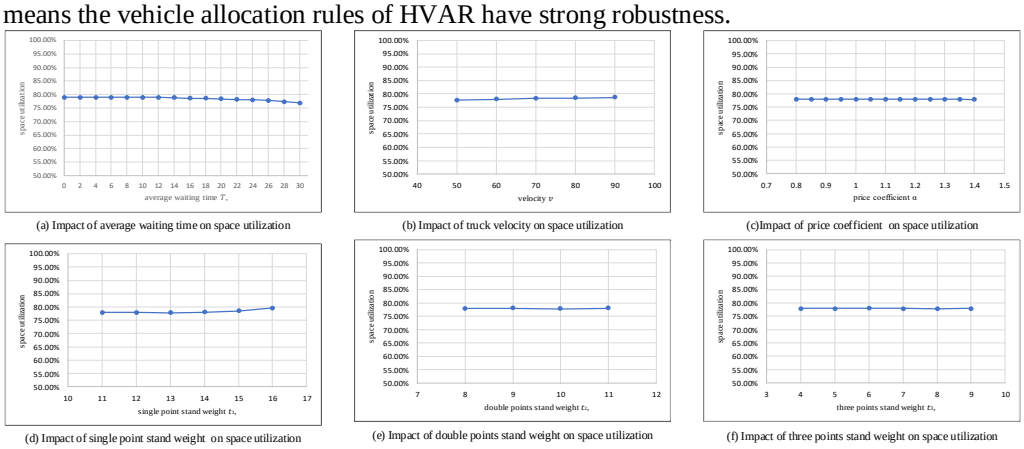
(a) Impact of average waiting time on cost savings (b) Impact of truck velocity on cost savings (c)Impact of price coefficient on cost savings

(d) Impact of single point stand weight on cost savings (e) Impact of double points stand weight on cost savings (f) Impact of three points stand weight on cost savings

Fig. 3 Influence of system parameters for cost saving

As shown in Fig. 3, the average waiting time T has a great influence on the transportation cost saving value. Especially when 10h < T < 20h, the influence gradually becomes prominent, and when T > 20h, the cost saving drops significantly. Therefore, the value of T should be carefully selected according to the actual situation. If possible, it should be set between 10 to 20 hours. The truck velocity v has a great relation with the cost saving value that the cost saving value increases with the acceleration of the speed. If possible, the value of v should be set as a larger value. Standard weight of three-point transport t_3 significantly influences the cost saving value and has a negative correlation, while other two standard weights t_1 and t_2 impact less. The freight rate α has a significant effect on the saving value.

Fig. 4 shows that the rate of space utilization is unrelated to the system parameters, which means the vehicle allocation rules of HVAR have strong robustness.



(a) Impact of average waiting time on space utilization (b) Impact of truck velocity on space utilization (c)Impact of price coefficient on space utilization

(d) Impact of single point stand weight on space utilization (e) Impact of double points stand weight on space utilization (f) Impact of three points stand weight on space utilization

Fig. 4 Influence of system parameters for space utilization

4.3 Joint optimization analysis on standard weight and freight rate

By the analysis in Section 4.2, the standard weights and freight rate are the core influence factors of transport cost. From the view of the enterprise, the most advantageous strategy is to reduce standard weights and freight rate simultaneously. However, it is unacceptable for the carrier. Therefore, by the game theory analysis, "one up and one down" should exist in the above two indicators, which can be accepted by the carriers in the game logic. There are two feasible optimization ideas: (i) Reduce at least one of the three standard weights and improve the freight rate; (ii) Raise at least one of the three standard weights while lower the freight rate.

We conducted a series of parameter combination optimization scheme tests. The freight rate α is selected within the range of ±5% of the current price (the step length is 1%). The value range of single-point standard weight t_1 is {11,12,13,14}, that of two-point standard t_2 is {8,9,10,11}, and that of three-point transport t_3 is {3,4,5,6}. Note that the selected value of the three standard weights should meet $t_1 > t_2 > t_3$.

A total of 828 preliminary schemes are obtained, from which several schemes with good performance on cost saving compose an alternative scheme table like Table 1. Then, provide the scheme table to the enterprise. Lastly, the enterprise and the carrier is able to coordinate to select the appropriate scheme. In fact, the selected scheme is not necessarily in the alternative scheme table. The enterprise is able to choose one of the preliminary schemes according to its actual demand, or coordinate with the carrier to customize multiple schemes and select the most favorable scheme through comparative analysis and verification of HVAR.

Table 1 Part of alternative schemes

t_1	t_2	t_3	v	T	α	Transport Cost of HVAR (￥)	Cost saving value (￥)	Saving rate
13	8	3	60	24	0.95	105824788	30091659	22.1398%
14	8	3	60	24	0.95	105873527	30042920	22.1040%
13	9	3	60	24	0.95	106589294	29327153	21.5773%
14	9	3	60	24	0.95	106640191	29276256	21.5399%
13	8	3	60	24	0.96	106938731	28977716	21.3202%
14	8	3	60	24	0.96	106987977	28928470	21.2840%
13	10	3	60	24	0.95	107390588	28525859	20.9878%
14	10	3	60	24	0.95	107416148	28500299	20.9690%
13	8	4	60	24	0.95	107444643	28471804	20.9480%

From the schemes with good cost saving effect in the 828 preliminary schemes, the scheme with higher cost saving value has a higher standard weight and lower freight rate, meanwhile, the lower the freight rate is, the more the cost saving will be. Hence, to ensure the maximal cost saving for the enterprise, the actual negotiation can be carried out according to the following steps: (i) The carrier determines the minimum acceptable freight rate; (ii) Negotiate the acceptable standard weight of three-point transport, which should be as low as possible; (iii) With the above two determined parameters, the scheme with large cost saving and the most increase of t_1 should be selected among the 828 preliminary schemes.

5 Conclusions and extensions

A heuristic vehicle allocation and routing planning algorithm (HVAR) to address the order batching and vehicle routing problem in cigarette distribution was presented. Focusing on total travelled cost minimization, HVAR reduces or even eliminates transportation subsidies due to small order quantities through order consolidation. At the same time, the vehicle allocation mechanism and the order splitting rules were proposed. The results of limited computational experiments show that the proposed algorithm can effectively reduce the logistics cost, and the vehicle allocation rules can effectively ensure the space utilization of the vehicle, which is extremely robust. Future work will be conducted to improve the proposed algorithm. Alternative

metaheuristic features will be examined to enhance batch generation while reducing computational cost.

References

1. S. Madankumar and C. Rajendran, "Mathematical models for green vehicle routing problems with pickup and delivery: A case of semiconductor supply chain," Computers and Operations Research, vol. 89, pp. 183-192 (2018).
2. S. Mancini, "The Hybrid Vehicle Routing Problem," Transportation Research Part C, vol. 78, pp. 1-12 (2017).
3. I. M. Coelho et al, " An ILS-based algorithm to solve a large-scale real heterogeneous fleet VRP with multi-trips and docking constraints," European Journal of Operational Research, vol. 250, (2), pp. 367-376 (2016).
4. D. Schubert, A. Scholz and G. Wäscher, "Integrated order picking and vehicle routing with due dates," OR Spectrum, vol. 40, (4), pp. 1109-1139 (2018).
5. M. Eskandarpour et al, "Enhanced multi-directional local search for the bi-objective heterogeneous vehicle routing problem with multiple driving ranges," European Journal of Operational Research, vol. 277, (2), pp. 479-491 (2019).

Abstracts

MISTA 2019

Exact and approximation algorithms for the expanding search problem

Ben Hermans · Jannik Matuschke · Roel Leus

1 Introduction

A searcher wants to find a certain object that has been hidden at one of the non-root vertices of a connected and undirected graph $G = (V \cup \{r\}, E)$ with root r. Vertex $v \in V$ has a probability p_v to contain the object, where $\sum_{v \in V} p_v = 1$, and each edge $e = \{v, w\} \in E$ has a length λ_e reflecting the distance (cost, time, or a composite measure) to move between vertices v and w. Given that the searcher can only probe one vertex at a time and that the search starts from the root r, the question is: how to search the graph's vertices in order to minimize the expected distance traveled before finding the object?

Under the paradigm of *expanding search*, introduced by Alpern & Lidbetter [1], the searcher can continue his search from every vertex that he has visited before. Such a search arises naturally when the cost to traverse an edge for a second time is negligible compared to the first traversal. When mining for coal, for example, moving within existing tunnels can occur in time negligible compared to the time needed to dig into a new site [1]. Similarly, when securing an area for a hidden explosive, moving within a secured area can occur considerably faster than securing a new location [2]. Another motivation arises from the setting where a disaster has destroyed certain links in a transportation network and repairing a link takes incomparably more time than traveling through a repaired link [3]. The goal is to find an expanding search, i.e. a sequence in which the vertices should be probed, that minimizes the expected distance traveled before finding the object.

Ben Hermans
Faculty of Economics and Business, KU Leuven, Belgium;
PhD Fellow, Research Foundation – Flanders (FWO)
E-mail: ben.hermans@kuleuven.be

Jannik Matuschke
Faculty of Economics and Business, KU Leuven, Belgium
E-mail: jannik.matuschke@kuleuven.be

Roel Leus
Faculty of Economics and Business, KU Leuven, Belgium
E-mail: roel.leus@kuleuven.be

In this talk, we describe a mixed integer program (MIP) to identify an optimal expanding search and strengthen the formulation with valid inequalities, which then leads to a branch-and-cut procedure. Next, we discuss how a greedy approach constitutes an 8-approximation for the problem.

2 Problem definition

Let $n = |V|$ be the number of non-root vertices, then an *expanding search* consists of a sequence of edges $\sigma = (e_1, \ldots, e_n)$ such that $r \in e_1$ and each edge e_k, with $k = 1, \ldots, n$, connects an unvisited vertex to one of the previously visited vertices. Hence, the edges $\{e_1, \ldots, e_k\}$ form a tree in G for every $k = 1, \ldots, n$. Next, denote by $\lambda(v, \sigma)$ the distance traveled until the search $\sigma = (e_1, \ldots, e_n)$ reaches vertex $v \in V$. That is, if e_k is the first edge in σ with $v \in e_k$, then $\lambda(v, \sigma) = \sum_{i=1}^{k} \lambda_{e_i}$. The expected distance traveled before finding the object under the expanding search σ then equals $c(\sigma) = \sum_{v \in V} p_v \lambda(v, \sigma)$. The *expanding search problem* consists of finding a sequence σ that minimizes this search cost. This problem is known to be NP-hard [3].

3 Results

In this talk, we first describe a mixed integer program for the expanding search problem that exploits the result that for a given tree, the optimal search sequence can be determined in polynomial time [1]. More particularly, we build upon the single-machine scheduling literature to obtain a linear program that identifies an optimal search sequence for a given tree. Since the edges of every expanding search form a spanning tree, the problem then reduces to identifying the 'optimal spanning tree'. Our MIP combines this selection of a tree with the linear program to find the optimal sequence on that tree. As such, we obtain a formulation that uncouples the selection of a tree (which is difficult) from sequencing on a given tree (which is easy). Finally, based on well-known results for the minimum spanning tree problem, we obtain valid inequalities that strengthen our formulation. Since there are exponentially many valid inequalities, including them all at once is undesirable. Therefore, we discuss an iterative algorithm that identifies a violated constraint if such constraint exists, which then leads to a branch-and-cut procedure.

In the second part of the talk, we discuss an approximation algorithm for the expanding search problem. Also here, we make use of existing solution methods for the special case of trees and, in particular, the concept of a graph's search density. The *search density* of a subgraph equals the ratio of (i) the total probability mass of all vertices in the subgraph and (ii) the total length of all edges in the subgraph. Alpern & Lidbetter [1] have shown that if G is a tree, there exists an optimal search that starts with searching the edges of a maximal search density subtree. Although this approach may be suboptimal in general graphs, we show that the resulting search cost is at most four times the optimal one. Contrary to the case of trees, however, finding a maximal search density subtree is strongly NP-hard in general graphs, and therefore it needs to be approximated itself. We describe a 1/2-approximation for this problem, which then results in an 8-approximation for the expanding search problem.

Table 1 Formulation [MIP$^+$] performs much better than formulation [MIP]. The CPU-time (in seconds) is averaged over the solved instances only.

	[MIP]		[MIP$^+$]	
n	# solved	CPU	# solved	CPU
10	10	0.18	10	0.16
20	10	1.91	10	0.90
30	10	17.64	10	4.30
40	6	189.79	10	10.36
50			10	27.64
60			10	100.10
70			8	110.08
80			7	248.30

Table 1 summarizes preliminary computational results for the mixed integer program without ([MIP]) and with ([MIP$^+$]) valid inequalities. For each graph size $n \in \{10, 20, \ldots, 80\}$, ten instances were generated randomly as follows. The length λ_e of each edge $e \in E$ was drawn uniformly from the set $\{1, 2, \ldots, 100\}$ and, to generate the probabilities, a weight a_v was drawn uniformly from the set $\{0, 1, 2 \ldots, 1000\}$ for each vertex $v \in V$, after which we set the probability $p_v = a_v / \sum_{w \in V} a_w$. We subsequently tried to solve these test instances using both formulations in CPLEX 12.8 with a time limit of 10 minutes. Our computational experiment shows that formulation [MIP$^+$] significantly outperforms formulation [MIP].

References

[1] S. Alpern and T. Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61(2): 265–279 (2013).

[2] S. Angelopoulos, C. Dürr, and T. Lidbetter. The expanding search ratio of a graph. *Discrete Applied Mathematics*, 260: 51 – 65 (2019).

[3] I. Averbakh and J. Pereira. The flowtime network construction problem. *IIE Transactions*, 44(8): 681 – 694 (2012).

MISTA 2019

Resource Balancing for Scheduling Problems

Masaki Itoi · Wei Wu · Atsuko Ikegami

1 Introduction

Resource balancing appears in various optimization problems, such as scheduling problems, assignment problems, and routing problems. In such problems, a decision maker looks for an optimal solution that minimizes the total cost while achieving fairness of distribution among the involved parties [1].

In this paper, we consider two models that aim to improve the tradeoff between total cost and fairness. We propose approaches to each model and then experimentally evaluate them, using a generalized assignment problem with resource balancing and a school timetabling problem for elementary schools in Japan [2].

2 Resource-balancing models for integer programming problems

Given sets $N = \{1, 2, \ldots, n\}$ and $M = \{1, 2, \ldots, m\}$, the integer programming (IP) problem with a non-negative integer variable x_j for each $j \in N$ is expressed as

$$\mathbf{P} \quad \min \quad \sum_{j \in N} c_j x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in N} a_{ij} x_j \geq b_i \qquad \forall i \in M, \tag{2}$$

$$x_j \in \mathbb{Z}_{\geq 0} \qquad \forall j \in N, \tag{3}$$

where $\boldsymbol{c}$ and $\boldsymbol{b}$ are vectors and A is a matrix. For convenience, we let X be the feasible region, i.e. $X = \{\boldsymbol{x} \mid \boldsymbol{x} \text{ satisfies (2)–(3)}\}$. We then consider a generalization of problem $\mathbf{P}$ in which x_j is required to be balanced for each $j \in \bar{N} = \{1, 2, \ldots, \bar{n}\}$ $(\subseteq N)$. In this paper, we consider two linear-scalarization models for this multi-objective optimization

M. Itoi · W. Wu · A. Ikegami
Seikei University, Japan
E-mail: dm186201@cc.seikei.ac.jp
E-mail: wuwei@st.seikei.ac.jp
E-mail: atsuko@st.seikei.ac.jp

problem:

$$\textbf{M1} \quad \min \ w_1 \sum_{j\in N} c_j x_j + w_2 u \qquad (4)$$
$$\text{s.t.} \ u \geq x_j \qquad j \in \bar{N}, \ (5)$$
$$\boldsymbol{x} \in X, \qquad (6)$$

$$\textbf{M2} \quad \min \ w_1 \sum_{j\in N} c_j x_j + w_2 \sum_{j\in \bar{N}} x_i^2 \qquad (7)$$
$$\text{s.t.} \ \boldsymbol{x} \in X, \qquad (8)$$

where u is a continuous variable and the non-negative weights w_1 and w_2 are the parameters of the scalarization provided by the decision maker. Model **M1** with a linear objective minimizes the maximum value of x_j for j in $\bar{N}$, while model **M2** with a quadratic objective function also minimizes the spread in an indirect way.

For model **M2**, we consider reformulations based on mixed integer programming (MIP) instead of solving it with a mixed integer quadratic programming (MIQP) solver. In the following models, $\hat{x} \in \mathbb{Z}_{\geq 0}$ and $\check{x} \in \mathbb{Z}_{\geq 0}$ are input parameters denoting an upper bound and lower bound, respectively, of the value of x_j for all j in $\bar{N}$. Our first MIP reformulation **M2-MIP**$(\hat{x}, \check{x})$ uses binary variables y_{jk} to represent x_j, i.e., $x_j = \sum_{k=\check{x}}^{\hat{x}} j y_{jk}$ and $\sum_{k=\check{x}}^{\hat{x}} y_{jk} = 1$, and the second term of the objective function in **M2** becomes $w_2 \sum_{j\in\bar{N}} \sum_{k=\check{x}}^{\hat{x}} k^2 y_{jk}$.

The other two models we considered use piecewise-linear functions with continuous variables y_{ik}:

$$\textbf{M2-U}(\hat{x}, \check{x}) \quad \min \ w_1 \sum_{j\in N} c_j x_j + w_2 \sum_{j\in\bar{N}} \left(\check{x}^2 + \sum_{k=\check{x}+1}^{\hat{x}} (2k-1) y_{jk} \right) \qquad (9)$$
$$\text{s.t.} \ \check{x} + \sum_{k=\check{x}+1}^{\hat{x}} y_{jk} = x_j \qquad \forall j \in \bar{N}, \qquad (10)$$
$$0 \leq y_{jk} \leq 1 \qquad \forall j \in \bar{N}, \forall k \in \{\check{x}+1, \check{x}+2, \ldots, \check{x}\}, \qquad (11)$$
$$\boldsymbol{x} \in X, \qquad (12)$$

$$\textbf{M2-L}(\hat{x}, \check{x}) \quad \min \ w_1 \sum_{j\in N} c_j x_j + w_2 \sum_{j\in\bar{N}} \left(\check{x}^2 + \sum_{k=\check{x}}^{\hat{x}} 2k y_{jk} \right) \qquad (13)$$
$$\text{s.t.} \ \check{x} + \sum_{k=\check{x}}^{\hat{x}} y_{jk} = x_j \qquad \forall j \in \bar{N}, \qquad (14)$$
$$0 \leq y_{jk} \leq 0.5 \qquad \forall j \in \bar{N}, \forall k \in \{\check{x}, \hat{x}\}, \qquad (15)$$
$$0 \leq y_{jk} \leq 1 \qquad \forall j \in \bar{N}, \forall k \in \{\check{x}+1, \check{x}+2, \ldots, \hat{x}-1\}, \qquad (16)$$
$$\boldsymbol{x} \in X. \qquad (17)$$

In **M2-U**, the piecewise-linear function is fitted to the quadratic function from above, while the one in **M2-L** is fitted from below.

Intuitively, we can assign 0 to $\check{x}$ and a sufficiently large value B to $\hat{x}$, and then any optimal solution of **M2-MIP**$(0, B)$, **M2-U**$(0, B)$ or **M2-L**$(0, B)$ is optimal for **M2**. However, tight setting of $\check{x}$ and $\hat{x}$ can reduce the number of variables. Because it is often easier in practice to solve the problem **P** than to solve **M2**, we propose a method of determining $\check{x}$ and $\hat{x}$ by solving problem **P** and problem $\min_{\boldsymbol{x}\in X} \sum_{j\in\bar{N}} x_j$ to obtain an optimal solution $\boldsymbol{x}^*$ and $\boldsymbol{x}^{**}$, respectively. Letting $\mu = \sum_{j\in\bar{N}} x_j^{**}$ and $\nu = \sqrt{w_1/w_2 \sum_{j\in N} c_j (x_j^{**} - x_j^*) + \sum_{j\in\bar{N}} (x_j^{**})^2}$, the following lemma holds.

Lemma 1 *Any optimal solution of* **M2-MIP**$(\check{x}, \hat{x})$, **M2-U**$(\check{x}, \hat{x})$ *and* **M2-L**$(\check{x}, \hat{x})$ *is optimal for* **M2** *when* $\check{x} = \left\lceil \max\left\{0, \frac{\mu - \sqrt{\mu^2 - \bar{n}\mu^2 + \bar{n}(\bar{n}-1)\nu^2}}{\bar{n}}\right\}\right\rceil$ *and* $\hat{x} = \left\lfloor \min\left\{\nu, \frac{\mu + \sqrt{\mu^2 - \bar{n}\mu^2 + \bar{n}(\bar{n}-1)\nu^2}}{\bar{n}}\right\}\right\rfloor$.

3 Iterative approach using piecewise-linear approximation

We additionally consider a heuristic technique in which we use piecewise-linear approximations to solve **M2**. In **M2-U** and **M2-L**, the length δ_{jk} of the kth piecewise interval of x_j is set to 1 or 0.5 according to (11), (15), and (16). However, a larger value of δ_{jk} or a narrower gap between $\check{x}$ and $\hat{x}$ can reduce the number of continuous variables needed. To improve the tradeoff between computational speed and approximation accuracy, we propose the use of an iterative approach in which we dynamically adjust $\check{x}$, $\hat{x}$, and δ_{jk} at each iteration. Given $\check{x}$, $\hat{x}$, and $\boldsymbol{\delta}$, the approach iteratively solves **M2-U** or **M2-L**, obtaining an optimal solution $\boldsymbol{x}^*$, and adjusting $\check{x}$, $\hat{x}$, and $\boldsymbol{\delta}$ for each x_j based on $\boldsymbol{x}^*$. For the **general M2-U** and the **general M2-L** (i.e., the case in which each piecewise interval can take any positive real value as long as continuity is maintained) using the values $\check{x}$ and $\hat{x}$ described in Lemma 1 for each element, the following property holds.

Lemma 2 *The optimal value z of the* **general M2-U** *(resp.,* **general M2-L***) is a valid upper bound (resp., lower bound) of the optimal value of* **M2**, *and* $z - 0.25\sum_{j\in\bar{N}} \max_k \delta_{ik}$ *(resp.,* $z + 0.25\sum_{j\in\bar{N}} \max_k \delta_{ik}$*) is a valid lower bound (resp., upper bound).*

4 Computational experiments

We tested our models and methods on a generalized assignment problem with resource balancing (GAP-RB). Given sets of tasks and agents, GAP-RB is solved to obtain an assignment requiring minimum total resource amounts s assigned such that e_i, the total resource amounts of tasks assigned to each agent i, are well balanced. We generated 45 instances based on the benchmark instances of the generalized assignment problem. The results showed that for most instances, models **M1** and **M2** obtained solutions with the same gaps between $\max_i e_i$ and $\min_i e_i$, and **M2** provided solutions with a lower value of s. For the model **M2**, algorithms using model **M2-U** or **M2-L** were faster than solving **M2** with an MIQP solver or solving **M2-MIP** with an MIP solver.

We then ran computational experiments to solve a school timetabling problem for elementary schools, based on real data from Japan. For this problem, Takahashi et al. proposed an MIP model that aims to minimize the violations of all soft constraints and promote fairness among classes [1]. They reported that an optimal solution could not be obtained within 10 days using the MIP solver Gurobi. However, with the algorithm using **M2-U**, we obtained a better solution in less than 16 hours.

References

1. Bertsimas, D., Farias, V.F., Trichakis, N.: The price of fairness. Operations research **59**(1), 17–31 (2011)
2. Takahashi, K., Loulenco, B., Akaike, Y., Yamaguchi, R., Yamamoto, T., Hayashida, S., Ikegami, A.: Timetabling for elementary schools. Transactions on Mathematical Modeling and Its Applications **10**, 80–91 (2017). [In Japanese]

MISTA 2019

Scheduling Tasks with Release and Delivery Times on Parallel Processors

Natalia Grigoreva

1 Introduction

We consider the problem of scheduling tasks with release dates and delivery times on parallel identical processors. Following the 3-field classification scheme proposed by Graham *et al.* [6], this problem is denoted by $P|r_j, q_j|C_{max}$. This problem plays a central role in some important applications [2], and it is NP-hard [12].

We consider a system of tasks $U = \{u_1, u_2, \ldots, u_n\}$. Each task is characterized by its execution time $t(u_i)$, its release time $r(u_i)$ the time at which the task is ready for processing and its delivery time $q(u_i)$.

Set of tasks is performed on m parallel identical processors. Any task can run on any processor and each processor can perform no more than one task at a time. Task preemption is not allowed. The schedule defines the start time $\tau(u_i)$ of each task $u_i \in U$. The makespan of the schedule S is the quantity

$$C_{\max} = \max\{\tau(u_i) + t(u_i) + q(u_i) | u_i \in U\}.$$

The objective is to minimize $C_{\max}$, the time by which all jobs are delivered. The problem is equivalent to model $P|r_j|L_{\max}$ with release times and due dates, rather than delivery times.

The problem $P|r_j, q_j|C_{\max}$ is a generalization of the one-machine scheduling problem with heads and tails $1|r_j, q_j|C_{\max}$. The problem arises as a strong relaxation of the multiprocessor flow shop problem [12]. The problem has been the subject of numerous papers. In fact, only Jackson's algorithm, which is a simple list scheduling, is used as a heuristic algorithm for the $P|r_j, q_j|C_{\max}$.

The worst-case performance of Jacksons algorithm has been investigated by Gusfield [9] and Carlier [3]. Gusfield [9] examined on-line polynomial time heuristic for the problem to minimize the maximum lateness of jobs with release times and due dates and proved that difference between the lateness given by the heuristic and the minimum possible lateness is bounded by $(2m-1)t_{max}/m$ and this bound is tight. Carlier [3] proved that $C_{\max} - C_{opt} \leq 2t_{\max} - 2$, where $C_{\max}$ is the

St. Petersburg State University
E-mail: n.s.grig@gmail.com

objective function of Jackson's rule schedule, and C_{opt} is the makespan of an optimal schedule.

Gharbi and Haouari [5]proposed improved Jacksons algorithm which use an O(n log n)-time preprocessing algorithm in order to reduce the number of jobs to be scheduled and investigated its worst-case performance.

An inserted idle time schedule (IIT) has been defined by J.Kanet and V.Sridharam [10] as a feasible schedule in which a processor is kept idle at a time when it could begin processing a task. In [7] we considered scheduling with inserted idle time for m parallel identical processors and proposed branch and bound algorithm for multiprocessor scheduling problem. In [8] we investigated the branch and bound algorithm for the problem with release times and due dates.

The goal of this paper is to propose an approximate IIT algorithm named MDT/IIT (maximum delivery time/ inserted idle time). and investigate its worst-case performance (section 2). To illustrate the effectiveness of this approach we tested it on randomly generated set of tasks. Section 3 contains a summary of this paper.

2 Approximation algorithm MDT/IIT

Algorithm MDT/IIT generates the schedule, in which processor is kept idle at the time when it could begin processing a job. Let S_k be the set of jobs that has been scheduled. Let $r_{\min} = \min\{r(i) \mid i \in U\}$ and $q_{\min} = \min\{q(i) \mid i \in U\}$.

Define the lower bound LB of the optimal makespan [3]:
$LB = \max\{r_{\min} + \sum_{i=1}^{n} t(i)/m + q_{\min}, \max\{r(i) + t(i) + q(i) \mid i \in U\}\}$.

Let $t_{\max} = \max\{t(i) \mid i \in U\}$. Let k tasks have been put in the schedule and partial schedule S_k have been constructed.

Let be $time_k[i]$ the time of the termination of the processor i after completion all its tasks. The approximation schedule is constructed by MDT/IIT algorithm as follows:

1. Determine the processor l_0 such as $t_{\min}(l_0) = \min\{time_k[i] | i \in 1..m\}$.
2. If there is no job u_i, such as $r(u_i) \leq t_{\min}(l_0)$ then
   $t_{\min}(l_0) := \min\{r(u_i) \mid u_i \notin S_k\}$.
   Select a job u with maximum delivery time
   $q(u) = \max\{q(u_i) \mid r(u_i) \leq t_{\min}(l_0)\}$.
3. If $t_{\min}(l_0) > t_{\max}$ then $v = u$; go to 10.
4. Select a job u^* with maximum delivery time, such as
   $q(u^*) = \max\{q(u_i) \mid t_{\min}(l_0) < r(u_i) < t_{\min}(l_0) + t(u)\}$.
5. If there is no such job u^* or one of inequality is hold
   $q(u) \geq q(u^*)$ or $q(u^*) \leq LB/3$, or $r(u^*) > t_{\max}$ then $v = u$. Go to 10.
6. Define the idle time of the processor before the start of job u^*
   $idle(u^*) = r(u^*) - t_{\min}(l_0)$
   If $q(u^*) - q(u) < idle(u^*)$, then we select job $v = u$. Go to 10.
7. Select a job u_1 which can be executed during the idle time of the processor without increasing the start time of the job u^*, namely such as
   $q(u_1) = \max\{q(u_i) \mid t_{\min}(l_0) \geq r(u_i)\ \&\ t(u_i) \leq idle(u^*)\}$.
   If we find u_1, then we select job $v = u_1$. Go to 10.

8. Select the ready job u_2 with maximum delivery time, such as $q(u_2) = \max\{q(u_i) \mid t_{\min}(l_0) < r(u_i)\ \&\ r(u_i) + t(u_i) \le r(u^*)\}$. If we find u_2, then we select job $v = u_2$. Go to 10.
9. We select the job $v = u^*$.
10. Define the start time of job v: $\tau(v) := \max\{t_{\min}(l_0), r(v)\}$; $time[l_0] := \tau(v) + t(v)$; $k := k + 1$. Include the job v in S_k.
11. If $k < n$, then go to 1.

We construct the approximation schedule $S = S_n$ and find the objective function $C_{\max}(S) = \max\{\tau(u_i) + t(u_i) + q(u_i) \mid u_i \in U\}$.

The algorithm sets on the processor l_0 the job u^* with the large delivery time $q(u^*)$. If this job is not ready, then the processor l_0 will be idle in the interval $[t_1, t_2]$, where $t_1 = time_k[l_0]$, $t_2 = r(u^*)$. In order to avoid too much idle of the processor the inequality $q(u^*) - q(u) \ge idle(u^*)$ is verified on step 6. If the inequality is hold, we choose job u^*. In order to use the idle time of the processor l_0 we select job u_1 or u_2 to perform in this interval. Job u^* starts at $\tau(u^*) = r(u^*)$.

Let algorithm generates a schedule S, then for each job j we have the start time $\tau(j)$. The objective function is $C_{\max}(S) = \max\{\tau(j) + t(j) + q(j) \mid j \in U\}$.

Let C_{opt} be the length of an optimal schedule.

Theorem 1 *$C_{\max}(S) - C_{opt} \le t_{\max}(2m-1)/m$. and this bound is tight.*

Example 1. Consider the $m^2 + m + 1$ jobs and m machine instance. There are $2m$ jobs $v_i : r(v_i) = 0$; $t(v_i) = m$; $q(v_i) = 0$. There are $m(m-1)$ jobs $u_i : r(u_i) = m - \delta$; $t(u_i) = 1$; $q(u_i) = m$ and job $a : r(a) = m - \delta$; $t(a) = m$; $q(a) = m$.

The makespan of MDT schedule is $C_{\max}(MDT) = -\delta + 5m - 1$. The optimal makespan is equal $3m$.

$C_{\max}(MDT) - C_{opt}$ is equil $2m - \delta - 1$, that is $2t_{max} - 1$.

The algorithms JR and MDT are in a certain sense opposites: if the algorithm JR generates a schedule with a large error, the algorithm MDT works well and vice versa. We propose the combined algorithm that builds two schedules: one schedule by the algorithm JR, the other by the algorithm MDT and selects the best.

To test the approximate MDT/IIT algorithm we conducted computational experiment. The quality of the solutions we estimated average ratio RT of the solution value over the lower bound of the makespan LB. We compared algorithms JR, MDT and the combined algorithm KA, that builds two schedules (one schedule by the algorithm JR, the other by the algorithm MDT) and selects the best solution. The combined algorithm allows to increase the number of optimal solutions by 5-9% and to improve RT.

3 Conclusion

We propose an approximate MDT/IIT for $P|r_j, q_j|C_{max}$ problem. We proved that $C_{\max}(S) - C_{opt} \le t_{\max}(2m-1)/m$, where $C_{\max}$ is the objective function of MDT/IIT schedule, and C_{opt} is the makespan of an optimal schedule. We propose a combined algorithm that builds two schedules (one schedule by the algorithm JR, the other by the algorithm MDT) and selects the best solution. Computational experiments have shown that the combined algorithm works better than each of the algorithms separately.

References

1. Baker, K.R.: Introduction to Sequencing and Scheduling. John Wiley & Son, New York (1974)
2. Brucker,P.: Scheduling Algorithms. fifth ed. Springer,Berlin (2007)
3. Carlier J. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. European Journal of Operational Research 1987;29:298306.
4. Gharbi A, Haouari M. Minimizing makespan on parallel machines subject to release dates and delivery times. Journal of Scheduling 2002;5:32955.
5. Gharbi A, Haouari M. An approximate decomposition algorithm for scheduling on parallel machines with heads and tails. Computers & Operations Research 34 (2007) 868883.
6. Graham, R.L., Lawner, E.L., Rinnoy Kan,A.H.G.: Optimization and approximation in deterministic sequencing and scheduling. A survey. Ann. of Disc. Math. **5** (10), 287–326 (1979)
7. Grigoreva, N.S.: Branch and bound method for scheduling precedence constrained tasks on parallel identical processors. Lecture Notes in Engineering and Computer Science: In proc. of The World Congress on Engineering 2014, WCE 2014, pp.832–836, London, U.K. (2014)
8. N.S.Grigoreva,*"Multiprocessor Scheduling with Inserted Idle Time to Minimize the Maximum Lateness"*, in Proceedings of the 7th Multidisciplinary International Conference of Scheduling:Theory and Applications. Prague,MISTA.2015, pp.814–816.
9. Gusfield D. Bounds for naive multiple machine scheduling with release times and deadlines. Journal of Algorithms 1984;5: 16.
10. Kanet, J.,Sridharan, V.: Scheduling with inserted idle time:problem taxonomy and literature review. Oper.Res. **48** (1), 99–110 (2000)
    Research Society of Japan ,**22**, 205—224 (1979)
11. Sourirajan K., Uzsoy R. : Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication. Jornal of Scheduling, **10**, 41—65 (2007)
12. J. Ullman. NP-complete scheduling problems *J. Comp. Sys. Sci.* **171**, pp.394-394, (1975).

MISTA 2019

Sequencing and Scheduling Appointments for Multi-stage Service Systems with Stochastic Service Durations and No-shows

Shenghai Zhou · Qing Yue

Due to the limited resources, appointment scheduling becomes an important problem in a large variety of service systems, especially in healthcare sector. For service industries, through careful appointment scheduling, the service providers can make full utilization of their working time, and the customers can be access to their required services without experiencing long waiting time. In appointment scheduling, one classic issue (see, for example, Mak et al. 2014b, Chen and Robinson 2014, Mancilla and Storer 2012) is to determine the start time of each customer in singe-stage service systems. In the issue, two-layer decision is needed to be made: sequencing jobs and scheduling appointments (In the rest of this paper, the terms "job" and "appointment" are used interchangeably). The former decision refers to determining the processing order of jobs and the later decision refers to determining the time interval (job allowance) to serve jobs for a given sequence. Both the above two types of decisions are common and important in service systems. For example, in healthcare sector, when there exists different types of patients in outpatient, or patients with different degree of emergency in operation rooms, the jobs sequencing decision is needed to be carefully made to improve the whole performance of the system. Besides, the waiting of patients and the idling of service providers are universal in service systems, to balance between them, scheduling appointments is another important affair to decision makers. In view of the importance of both sequencing jobs and scheduling appointments in service systems, the joint sequencing and scheduling appointments problems have been attracted much attention from practitioners and academia.

The vast majority of studies on sequencing and scheduling appointments problems in literature focus on single-stage service systems, which involve only one service process. In single-stage service systems, service durations are usually stochastic due to different requirements of customers. As a result, a common examined objective in sequencing and scheduling appointments problems is to minimize the total expected costs incurred by customers' waiting times and service provider's idling times (Mak

First Author
School of Business Administration, Jiangxi University of Finance and Economics
E-mail: zhoushenghai@jxufe.edu.cn

Second Author
Management School, Shanghai University of International Business and Economics
E-mail: yueqing@suibe.edu.cn

et al. 2014b). However, in some practical service systems, customers may go through multiple service processes. For example, when having childhood vaccinations, children have to go through the registration and pre-examination (like body temperature measurement), and the vaccinations. For the clinic, both jobs sequencing and appointments scheduling are critical to the system performance. Therefore, it is valuable to investigate sequencing and scheduling appointments problems in multi-stage service systems. And a nature optimization problem is to jointly sequence jobs and schedule appointments, so as to minimize total expected costs associated with customers' waiting times and service providers' idle times over multiple stages.

In this study, we consider a joint sequencing and scheduling appointments problem with stochastic service times and no-shows in multi-stage service systems. In the problem, the decision maker needs to determine: (1) the processing sequence of jobs; and (2) for a given sequence, the job allowance for each customer in the first stage, so as to minimize the total expected weighted costs for customers' waiting times and service providers' idle times over multiple stages. Here, we assume that customers just need to make appointments in the first stage, and then go through the rest stages in a First Come First Served (FCFS) rule. The phenomenon is common in reality, and also can be illustrated by literature on multi-stage appointment scheduling (Zhou and Yue 2019).

There are few studies investigate multi-stage appointment scheduling problems in literature. And those studies often assume that all jobs go through multiple stages according to a fixed sequence (Azadeh et al. 2015, Castro and Petrovic 2012, Pérez et al. 2013, Klassen and Yoogalingam 2018). For example, Castro and Petrovic (2012) study a multi-stage radiotherapy pretreatment scheduling problem. They assume that the sequence of operations for each patient is known and formulate a deterministic integer program. Pérez et al. (2013) study an online patient and resource scheduling problem in nuclear medicine departments and assume a predetermined sequence for each type of patients. For the problem, they formulate a two-stage stochastic program. For the number of severs in each stage in a multi-stage service system, most works, including ours, consider the case with only a single service provider at each stage (Azadeh et al. 2015, Castro and Petrovic 2012, Kazemian et al. 2017, Zhou and Yue 2019), and only few works consider the case with multiple service providers at each stage (Wang et al. 2018). To solve multi-stage appointment scheduling problems, some works in literature employ a stochastic optimization approach to model problems and solve them through simulation methods (Saremi et al. 2013, Kazemian et al. 2017, Zhou and Yue 2019), and some works analyze the problems with the framework of queueing and obtain some effective schedule policies (Kuiper and Mandjes 2015, Wang et al. 2018, 2019). In this study, the former approach is applied. Two major characteristics can distinguish our work from these above studies on multi-stage appointment scheduling, namely the decision making and the uncertainties in service systems. First, only the decision of scheduling jobs is needed to be determined in those studies, whereas both the sequencing and the scheduling appointments are needed to be decided in our study. Second, those studies only consider stochastic service durations, whereas we take both stochastic service durations and customer no-shows into consideration.

For our studied multi-stage sequencing and scheduling appointments problem, it is a tough work to obtain the exact optimal solution, and the challenges come from three aspects. First, job sequencing decision seems to be extremely difficult to be made for sequencing and scheduling appointments problems (Mak et al. 2014a), even for single-stage service systems (Denton et al. 2007, Gupta 2007, Mancilla and Storer

2012). For example, Mancilla and Storer (2012) prove such scenario based problem is NP-completed. Thus, existing studies on single-stage sequencing and scheduling appointments problems focus either on developing effective heuristics (Denton et al. 2007, Gupta 2007, Mancilla and Storer 2012), or solving approximated problems (Mak et al. 2014b). Second, the multi-stage service system is much more complicated than the single-stage service system, because performance indicators (e.g., waiting times and idle times) among different stages are interdependent and interactive. This leads to more difficulties in problem analysis. Third, stochastic service durations and no-shows also bring some obstacles in solving the problem.

Due to the difficulties of solving our studied multi-stage sequencing and scheduling appointments problem, we try to find near optimal solutions through studying the structure characteristics of the problem. Specifically, we first formulate it as a stochastic program, and exploit the sample average approximation (SAA) approach to reformulate it as a mixed integer linear program in further. Then we transform the stochastic program int a two-stage optimization problem and develop a standard Benders decomposition algorithm. Due to the long running time of Bender decomposition algorithm, we simplify the master problem in the algorithm and propose a Benders decomposition based algorithm. Finally, we propose some easy-to-implement heuristics and conduct computational experiments to evaluate the performance of developed Benders decomposition based algorithm and the proposed heuristics.

Our study mainly show that: (1) Our developed Benders decomposition based algorithm performs well compared with the benchmark which solves the DLP through optimization software directly. (2) The proposed easy-to-implement heuristics are competent.

In view of literature, we make the following contributions to the emerging multi-stage appointment scheduling literature. First, compared with existing works on appointment scheduling, we extend the work in Mak et al. (2014b) into multi-stage service systems, and make an extension of the work in Zhou and Yue (2019) by considering the decision of sequencing jobs and customer no-shows. To the best of our knowledge, we are the first to study the sequencing and scheduling appointments problem in multi-stage service systems, and take customer no-shows into consideration. Second, from a methodological point of view, we develop an effective algorithm based on Benders decomposition to achieve near optimal solution.

References

Azadeh A, Baghersad M, Farahani MH, Zarrin M (2015) Semi-online patient scheduling in pathology laboratories. *Artificial Intelligence in Medicine* 64(3):217–226.

Castro E, Petrovic S (2012) Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. *Journal of Scheduling* 15(3):333–346.

Chen RR, Robinson LW (2014) Sequencing and scheduling appointments with potential call-in patients. *Production and Operations Management* 23(9):1522–1538.

Denton B, Viapiano J, Vogl A (2007) Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health Care Management Science* 10(1):13–24.

Gupta D (2007) Surgical suites' operations management. *Production and Operations Management* 16(6):689–700.

Kazemian P, Sir MY, Van Oyen MP, Lovely JK, Larson DW, Pasupathy KS (2017) Coordinating clinic and surgery appointments to meet access service levels for elective surgery. *Journal of biomedical informatics* 66:105–115.

Klassen KJ, Yoogalingam R (2018) Appointment scheduling in multi-stage outpatient clinics. *Health Care Management Science* 1–16.

Kuiper A, Mandjes M (2015) Appointment scheduling in tandem-type service systems. *Omega* 57:145–156.

Mak HY, Rong Y, Zhang J (2014a) Appointment scheduling with limited distributional information. *Management Science* 61(2):316–334.

Mak HY, Rong Y, Zhang J (2014b) Sequencing appointments for service systems using inventory approximations. *Manufacturing & Service Operations Management* 16(2):251–262.

Mancilla C, Storer R (2012) A sample average approximation approach to stochastic appointment sequencing and scheduling. *IIE Transactions* 44(8):655–670.

Pérez E, Ntaimo L, Malavé CO, Bailey C, McCormack P (2013) Stochastic online appointment scheduling of multi-step sequential procedures in nuclear medicine. *Health Care Management Science* 16(4):281–299.

Saremi A, Jula P, ElMekkawy T, Wang GG (2013) Appointment scheduling of outpatient surgical services in a multistage operating room department. *International Journal of Production Economics* 141(2):646–658.

Wang D, Morrice DJ, Muthuraman K, Bard JF, Leykum LK, Noorily SH (2018) Coordinated scheduling for a multi-server network in outpatient pre-operative care. *Production and Operations Management* 27(3):458–479.

Wang D, Muthuraman K, Morrice D (2019) Coordinated patient appointment scheduling for a multistation healthcare network. *Operations Research* 67(3):599–618.

Zhou S, Yue Q (2019) Appointment scheduling for multi-stage sequential service systems with stochastic service durations. *Computers & Operations Research* .

MISTA 2019

Dealing with uncertainty in time-relaxed round robin sports timetabling

Xiajie Yi · Dries Goossens

1 Introduction

This study focuses on uncertainty in time-relaxed round robin sports timetabling, and how to mitigate its effects using combined proactive and reactive approaches. Sports timetables determine when team pairs will play against each other and in which venue, and can be divided into two types: temporally-constrained and temporally-relaxed timetables. The first type is also known as compact timetables, using the minimum number of time slots required to play all games. Note that although multiple matches can be played on the same time slot, each team can play at most once on a slot. Nowadays, all European professional football timetables are compact [1]. In contrast, the number of time slots is larger than the minimum number of time slots needed in temporally-relaxed round robin timetables (and no team plays in each slot). Cricket [2,3], hockey [4] and non-professional indoor football [5] can be viewed as examples of sports with temporally-relaxed timetables.

Despite the hard efforts invested at the beginning of the season, timetables are rarely played as initially planned. During the tournament, games can be postponed because of unexpected events such as bad weather conditions, conflicts with other competitions, etc. Therefore, it is necessary to have other available time slots for those games. In compact timetables, inserting so-called catch-up slots into the initial timetable has been applied in order to create buffers for rescheduling postponed games [7]. However, a postponed game can be only rescheduled to a catch-up slot if the associated teams have no game scheduled on it. Fixtures as they were effectively played are referred to as the realized timetable.

In time-relaxed timetables, there are already (many) more time slots than required available. However, not every time slot is equally suitable for accommodating a post-

Xiajie Yi
Department of Business Informatics and Operations Management, Faculty of Economics and Business Administration, Ghent University, Belgium
E-mail: xiajie.yi@ugent.com

Dries Goossens
Department of Business Informatics and Operations Management, Faculty of Economics and Business Administration, Ghent University, Belgium
E-mail: dries.goossens@ugent.be

poned game. Assume that we have a set of teams $T = \{1, 2, ..., n\}$, playing a double-round robin time-relaxed tournament in a set of time slots $S = \{1, 2, ..., |S|\}$. Each team $i \in T$ provides a *home game set* ($H_i \subseteq S$) with time slots on which their home venue is available, and a *forbidden game set* ($F_i \subseteq S$, with $B_i = |F_i|$) with time slots on which no game can be played. Notice that for each team i, we assume $H_i \cap F_i = \emptyset$ since teams will not be able to organize a home game if their players are not available. Since every team has to play home $n-1$ times, implying that every team must have its home venue available on at least $n-1$ slots, we assume that $|H_i| \geqslant n-1$. Thus, every team has $A_i = |H_i| - (n-1)$ *extra home slots* [5]. Time-relaxed timetabling is a problem that is relevant in practice, and that it occurs in e.g. amateur indoor football (see e.g. [5]) and amateur table tennis leagues (see e.g. [6]).

To evaluate the quality of a time-relaxed round robin timetable, Suksompong [8] defines the *games played difference index* (GPDI) as the minimum integer p such that the difference between the number of games played by any two teams is at most p at any point in the timetable. For a compact timetable with an even number of teams, the $GPDI$ is 0, and $GPDI$ is 1 for a compact timetable with a odd number of teams. A timetable with a low GPDI is desirable as it results in more accurate tournament rankings and it may also reduce the opportunities for match fixing. Thus, we are aiming at designing such timetables and maintaining them despite unforeseen events.

2 Methodology

To develop timetables for time-relaxed round robin tournaments that can handle postponed games without much impact on the GPDI, we propose combined proactive and reactive strategies. The proactive approaches focus on developing an initial schedule that anticipates the realization of unpredicted events during the season. Reactive policies, on the other hand, revise or re-optimise the initial timetable after the unforeseen event has occurred.

We call a time slot on which only one team has its venue available for playing an *isolated slot*; all other teams do not have this slot in their home game sets. Since teams that already play on a slot are not available for a postponed game on that slot, and since home game slots are relatively scarce, isolated slots increase the number of opportunities to reschedule games. Hence, the proactive policy is to generate an initial timetable for which the GPDI is minimal, while maximzing the number of isolated slots that are not used in the initial timetable. There are various ways in which a timetable that measures GPDI can be computed; Van Bulck *et al.* [9] for instance discuss a fix-and-relax approach.

The reactive policies concentrate on rescheduling a postponed game to an appropriate available slot, however, taking into account that no slot can have more than one match involving the same team. Note that we do not allow to make changes to other matches from the initial timetable. We also assume that a postponed game can only be rescheduled to a slot which is scheduled after its original slot. We consider two reactive policies: (i) rescheduling a postponed game to the first available slot (PA); (ii) rescheduling a postponed game to the best available slot with respect to the GPDI (PG).

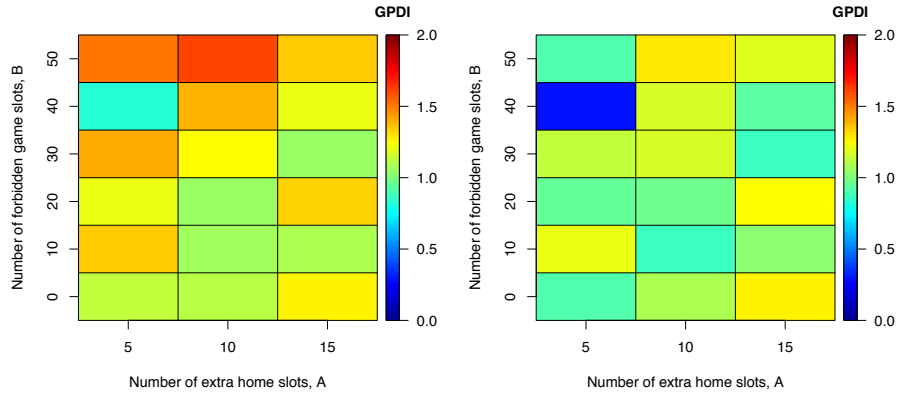
3 Simulation and discussion

We test our approach on a double round robin tournament with 14 teams and 100 slots. Each team is assumed to have a same size of home game set and forbidden game set, so we have $A_i = A, B_i = B, \forall i \in T$. Characteristics of the initial schedule are shown in Table 1 for various values of A and B. The table shows that the number of isolated slots first slightly increases, and then drops when having more extra home slots. This decrease occurs because it is difficult to avoid overlap between home slots for different teams if the number of extra home slots is relatively large.

Table 1 GPDI / unused isolated slots for the initial timetable for various values of A and B

	B					
A	0	10	20	30	40	50
5	2/12	2/12	2/12	2/12	2/12	3/12
10	2/13	2/13	2/13	2/13	2/13	3/13
15	2/3	2/3	2/3	2/3	2/3	2/3

In terms of the GPDI, the PG policy (Figure 1(b)) deals with postponed games better than PA policy (Figure 1(a)). The GPDI increases with the number of slots in the forbidden game set, B. However, the value of the GPDI does not always grow if there are more extra home slots, A. The reason behind this can be related to the position of the home slots and the position of postponed games, and requires further research. The difference on the growth in GPDI between the reactive polices is shown in Figure 2. It is interesting that the difference on the growth of GPDI increases with the number of slots in the forbidden set, but it is not impacted by the number of extra home slots.



(a) PA (b) PG

Fig. 1 Average increase in GPDI between initial and realized timetables for PA and PG.

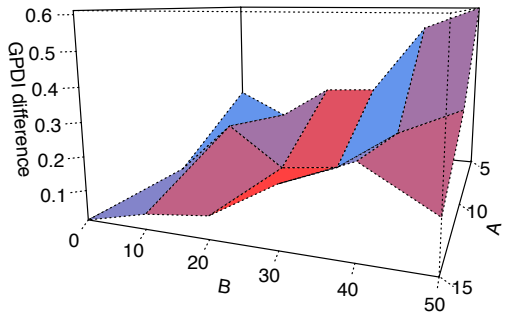


Fig. 2 Differences on the average increase in GPDI between policies PA and PG.

References

1. Goossens, D. & Spieksma, F., Soccer schedules in Europe: an overview, Journal of Scheduling,15(5): 641-651(2011).
2. Armstrong, J. & Willis,R., Scheduling the cricket world cup - a case study, Journal of the Operational Research Society, 44: 1067-1072 (1993).
3. Willis, R. & Terrill, B., Scheduling the Australian state cricket season using simulated annealing, Journal of the Operational Research Society 45: 276-280 (1994).
4. Costa, D., An evolutionary tabu search algorithm and the NHL scheduling problem, INFOR 33: 161-178 (1995).
5. Van Bulck, D., Goossens, D. R., & Spieksma, F. C., Scheduling a non-professional indoor football league: a tabu search based approach. Annals of Operations Research, 275(2), 715-730 (2019).
6. Schönberger, J., Mattfeld, D. C., & Kopfer, H. Memetic algorithm timetabling for non-commercial sport leagues. European Journal of Operational Research, 153(1), 102-116 (2004).
7. Yi, X., Goossens, D., & Nobibon, F., Proactive and reactive strategies for scheduling soccer leagues based on the empirical analysis of the quality of soccer league schedules. European journal of operational research, doi: 10.1016/j.ejor.2019.09.038 (2019).
8. Suksompong, W., Scheduling asynchronous slot-robin tournaments. Operations Research Letters, 44(1), 96-100 (2016).
9. Van Bulck, D. & Goossens, D., Scheduling time-relaxed double round-robin tournaments with availability constraints. In 51st Annual Convention of Operational Research Society of India & International Conference (ORSI 2018), Mumbai, India. pp. 73-73.

MISTA 2019

A Novel Artificial and Heuristic Hybrid Algorithm Method for Satellite Data Downlink Scheduling Problem with Malfunctions

Zhongshan Zhang • Benjie Pi • Yanjie Song • Lining Xing • Yingwu Chen

Data downlink scheduling of satellite is an important part of ensuring normal operation of satellites and successful completion of imaging acquiring tasks. It is also a key issue in satellite planning areas. During the operation, normal operation and execution tasks of satellites may be affected due to various faults. Satellite faults can result in data downlink failures, and task execution scheme need to be re-defined to minimize the impact. This paper proposes a satellite data downlink scheduling problem considering satellite malfunction. An in-orbit rescheduling framework for satellite is proposed which uses BP Network and a heuristic algorithm. The rescheduling framework uses artificial intelligence to predict satellite failures in advance and replan tasks. BP Network is used as a method of predicting satellite malfunctions and inject the model into satellite after offline model training. Satellite uses BP Network to perform online fault prediction. According to the prediction result, the heuristic algorithm is used to replan satellite downlink tasks. A new satellite mission execution plan after fault recovery is given by using this framework. In this paper, a variety of simulation experiments are designed. From experimental results, it can be proved that the proposed in-orbit rescheduling method can effectively solve this problem.

Zhongshan Zhang
College of Systems Engineering, National University of Defense Technology
E-mail: zszhang@nudt.edu.cn

Benjie Pi
LEOBIT Technology Co.Ltd.
E-mail: pibenjie@leobit.com

Yanjie Song
College of Systems Engineering, National University of Defense Technology
E-mail: songyj_2017@163.cn

Lining Xing
College of Systems Engineering, National University of Defense Technology
E-mail: xing2999@qq.com

Yingwu Chen
College of Systems Engineering, National University of Defense Technology
E-mail: ywchen@nudt.edu.cn

MISTA 2019

EURO-2019 conference scheduling
- Use of session patterns

Thomas Stidsen

1 Introduction to EURO conference scheduling

Scheduling large conferences is complicated. Many choices needs to be made and for large conferences this creates a hard planning problem, to be handled by the program chair under serious time pressure. This article is about the planning of the EURO-2019 conference in Dublin, 23/6-26/6 2019. The approach is tailored to the EURO/IFORS conferences, which we have previously planned 2015, 2016, 2017 and 2018, see [1].

EURO is the second largest Operations Research conference in the world. In 2019 there were app. 2500 participants, 2049 presentations, 545 sessions distributed among 123 streams, grouped into 25 Areas. Each stream has one or two organisers leading to about 200 stream organisers. Each stream has a number of presentations and the number of sessions for the stream is then the number of streams divided by 4, rounded up. To create the program there are two main tasks:

- To decide for each session of each stream, which timeslot it should have and in which room. This is the planning problem we are considering in this article. Professor Luis Gouveia from University of Lisbon was the Program Chair of the conference and the one ultimately responsible for the final program, based on the plan presented in this abstract.
- To decide for each stream how the presentations are distributed among the different session. This is handled by the stream organisers.

2 What is a good conference schedule

The aim of the conference schedule we produce is to facilitate a good conference for the participants. The aim when creating the conference schedule is then to schedule the sessions such that as many participants as possible can participate in the sessions they want: They should not be overlapping and the session they attend should be close together. Also we would like the sessions of a stream to take place in the same room, to make navigation easier for the participants and we would like the sessions to be contiguous.

Given a set of sessions to allocate, there are a number of hard constraints to satisfy:

- All sessions should be allocated to a timeslot and a room.
- Two sessions of a stream cannot be allocated to the same timeslot.

Thomas Stidsen
Technical University of Denmark
E-mail: thst@dtu.dk

– At most one session can be allocated to each room in each timeslot.
– Sessions can only be allocated to rooms with a sufficient no of seats.

We solve the model lexicographically, first grouping streams together according to which areas they are part of (each stream is part of an overall area/theme) and afterwards we minimize the overlap of similar streams. In these objectives we ignore the problem of getting contiguous streams and the problem of allocating the sessions of the same stream to the same rooms. This is possible since we choose a pattern based approach.

3 Stream Patterns

Instead of using individual binary variables for each session in each stream, we use patterns to specify that when choosing this pattern, then the sessions of that stream are allocated to specific timeslots, see the table below.

Pat	MA	MB	MC	MD	TA	TB	TC	TD	WA	WB	WC	WD
1	X	X	X	X	X	X	X	X				
2		X	X	X	X	X	X	X	X			
3			X	X	X	X	X	X	X	X		
4				X	X	X	X	X	X	X	X	
5					X	X	X	X	X	X	X	X

For each stream we generate all the contiguous patterns possible, as shown in the table for a stream with 8 sessions. The patterns are saved as an incidence matrix for each stream: $PAT_t^p \in \{0,1\}$ which is 1 if there is a session of pattern p in timeslot t. The basic decision variable in our MIP models is then $x_r^{s,p} \in \{0,1\}$, which is 1 if pattern p is used for stream s allocating sessions in room r. By using this formulation it is ensured that all sessions are allocated to only one room and (almost) contiguous streams.

4 Special constraints

Given the size of the EURO conference, there are always special requests like: For a certain stream, no sessions should be allocated on Tuesday or for a certain stream there has to be at least 2 sessions of that stream on Wednesday. To cater this demand, we allow the program chair to formulate special constraints of the type:

$$Low \leq \sum_t SC_t^{sc,s} \cdot PAT_t^p \cdot x_r^{s,p} \leq Up \ \forall sc, s, p$$

Where the $SC_t^{sc,s}$ is the incidence matrix of special constraint sc and PAT_t^p is (one) of the patterns for a stream. This approach is rather flexible and many different special requirements can be handled this way, at EURO 2018 there were 60 of these special requirement constraints. The special constraints do however create a new problem: It might be that the special requirements deem **all** the possible patterns for a stream infeasible. If this is not handled, the whole model becomes infeasible.

4.1 Generating feasible patterns

It is simple to check for each stream if at least one of the contiguous patterns are feasible. If that is not the case we need to generate at least one feasible pattern. This is done with the MIP model below, which minimize the number of "holes" in the pattern, still satisfying all the special requirements:

$$
\begin{aligned}
\min_s \quad & Last - First \\
\text{s.t.} \quad & \\
& \sum_t y_t = NO_SESSIONS \\
& Low \leq \sum_t SC_t^{sc,s} \cdot y_t \leq Up \qquad \forall\ sc \\
& First \leq t * y_t + T \cdot (1 - y_t) \qquad \forall t \\
& Last \geq t * y_t \qquad \forall t \\
& y_t \in \{0,1\},\ First, Last \in Z^+
\end{aligned}
$$

The above model returns a compact stream pattern, unless it is infeasible, but then no feasible allocation of the sessions exists, and the program chair will have to re-consider the special constraints. It is important to note that solving these models is very fast, due to the low number of binary variables.

5 Data implementation and results

All the data processing and model building was done in the Julia programming language, utilizing the Jump mathematical programming package. All data was contained in one excel file, including the special constraint data. Keeping the data source simple was crucial since the planning team was situated in Lisbon, Bruxelles, Copenhagen and Dublin. The Gurobi solver was used and the models could always be solved in less than 5 minutes. In Dublin, there was 10 different buildings and the Dublin organisers had provided a walking-time matrix. In step 1 of the optimization it was however possible to place each area in one building only, leaving a walking time objective of 0. In step 2 the overlap of different related stream groups was minimized and the in optimal solution there was a total overlap of 10 sessions.

6 Future development and conclusion

The presented approach is the result of the planning of 5 different EURO/IFORS conferences and the approach has produced satisfactory results. In the future we would like to get model the overlapping of related streams better, probably using an oriented graph and we would like to get better estimates of the number of attendees to the streams.

References

1. Thomas Stidsen, David Pisinger, and Daniele Vigo. Scheduling euro-k conferences. *European Journal of Operational Research*, 2017.

MISTA 2019

Scheduling a single batch machine with non-identical job sizes and incompatible job families

Fan Yang · Morteza Davari · Wenchao Wei · Roel Leus

1 Introduction

We study the problem of scheduling jobs on a single batch machine with non-identical job sizes and incompatible job families. In practice, the job size will typically reflect the physical size, or the number of units of a product in a job. In burn-in operations of semiconductor production, for example, a job corresponds to one or more product-specific boards each containing a number of integrated circuits, to be loaded into an oven. The size of the job is determined by the number of boards it requires, and the capacity of the machine (the oven) is expressed in terms of the number of boards it can hold (Uzsoy, 1994). A job family consists of all jobs that have the same processing recipe. In steel heat treatment, for instance, different jobs may have different cooling requirements, in which case they cannot be processed simultaneously (Dobson and Nambimadom, 2001). Jobs from different families are thus incompatible: they cannot be combined in the same batch.

Dobson and Nambimadom (1992, 2001) first proposed this problem on a single machine to minimize total weighted completion time. They showed that it is NP-hard, and gave a mixed-integer formulation and a lower bound. A special case can be solved in polynomial time and three heuristics are proposed for general cases. For the same problem setting, Azizoglu and Webster (2001) provided a branch-and-bound algorithm that can solve up to 25 jobs in reasonable runtime.

Fan Yang
Research Centre for Operations Research and Business Statistics, KU Leuven, Leuven, Belgium
E-mail: f.yang@kuleuven.be

Morteza Davari
Research Centre for Operations Management, KU Leuven, Brussels, Belgium
E-mail: morteza.davari@kuleuven.be

Wenchao Wei
School of Economics and Management, Beijing Jiaotong University, Beijing, China
E-mail: weiwenchao@bjtu.edu.cn

Roel Leus
Research Centre for Operations Research and Business Statistics, KU Leuven, Leuven, Belgium
E-mail: roel.leus@kuleuven.be

2 Problem statement

We schedule a set $N = \{1, \ldots, n\}$ of n independent jobs partitioned in m job families. Each family j is in the set $\mathcal{F} = \{1, \ldots, m\}$, and coincides with a set of jobs $F_j \subseteq N$, where $\cup_{j=1}^{m} F_j = N$ and $F_i \cap F_j = \emptyset$ for all $\{i, j\} \subset \mathcal{F}$. Each job i has a weight w_i and a size v_i, and jobs in the same family j have the same processing time p_j. There is a single batch processing machine with capacity V, which can process several jobs from the same family simultaneously, but jobs from different families cannot be processed together. We assume $0 \leq v_i \leq V$ for all $i \in N$. The total size of the jobs in one batch (one set of jobs processed together) cannot exceed the machine capacity. The processing time μ_k of a batch k containing family j equals p_j. Once the processing of a batch starts, there is no interruption and other jobs cannot be added to the batch before completion. A job cannot be split across batches. The objective is to minimize the total weighted completion time of the jobs; the completion time of all jobs in a batch is the completion time of the batch.

Below we describe an assignment-based formulation based on Dobson and Nambimadom (2001). In this formulation, we work with a set of *potential* batches $\Omega = \{1, \ldots, n\}$; each batch can be assigned to one of the positions in set $\mathcal{L} = \{1, \ldots, n\}$ in the processing sequence. Family $j \in \mathcal{F}$ is associated with the set B_j containing $|F_j|$ potential batches for jobs in the family. These B_j constitute a partition of Ω, and for ease of indexing we associate batch $k \in \Omega$ with the family $j \in \mathcal{F}$ which has job $k \in F_j$. The binary decision variables x_{ik} are defined as follows: $x_{ik} = 1$ if job i is assigned to batch k and $x_{ik} = 0$ otherwise, for $i \in N$ and $k \in B_j$ such that $i \in F_j$. We also use the binary variables y_{kl}, which equal 1 if batch k is sequenced at the lth position and $y_{kl} = 0$ otherwise, for $k \in \Omega$ and $l \in \mathcal{L}$. The variables BC_l and C_i are non-negative and represent the completion time of the lth batch and of job i, respectively.

$$\min \sum_{i=1}^{n} w_i C_i \tag{1}$$

subject to:

$$\sum_{i \in F_j} v_i x_{ik} \leq V, \qquad \forall k \in \Omega \text{ with } k \in B_j, \tag{2}$$

$$\sum_{k \in B_j} x_{ik} = 1, \qquad \forall j \in \mathcal{F}, \forall i \in F_j, \tag{3}$$

$$\sum_{l \in \mathcal{L}} y_{kl} \leq 1, \qquad \forall k \in \Omega, \tag{4}$$

$$\sum_{k \in \Omega} y_{kl} \leq 1, \qquad \forall l \in \mathcal{L}, \tag{5}$$

$$x_{ik} \leq \sum_{l \in \mathcal{L}} y_{kl}, \qquad \forall i \in N, \forall k \in B_j \text{ with } i \in F_j, \tag{6}$$

$$BC_1 = \sum_{k \in \Omega} \mu_k y_{k1}, \tag{7}$$

$$BC_l = BC_{l-1} + \sum_{k \in \Omega} \mu_k y_{kl}, \qquad l \in \mathcal{L} \backslash \{1\}, \tag{8}$$

$$C_i \geq BC_l - M(2 - x_{ik} - y_{kl}), \qquad \forall i \in N, \forall k \in B_j \text{ with } i \in F_j, \forall l \in \mathcal{L}. \tag{9}$$

The objective is to minimize the total weighted completion time. Constraints (2) impose that the total size of the jobs in batch k cannot exceed the machine capacity V. Constraints (3) ensure that each job i is assigned to one batch k. Constraints (4) and (5) establish that each batch is scheduled at most once. Constraints (6) make sure that each non-empty batch is scheduled. Constraints (7) to (9) determine the completion time of the lth batch and of job i, respectively, where M is a large number. It can be seen that this formulation contains symmetry via the choice of empty batches; this can be remedied using symmetry-breaking constraints.

3 Branch-and-Benders-cut

In our branch-and-Benders-cut (B&BC) algorithm, we decompose the original problem into a master problem for job batching and a subproblem for batch scheduling. The subproblem is called each time when a new job partition is found in the master problem, which then generates a Benders cut.

The master problem takes the following form:

$$\min \eta \tag{10}$$

subject to:

$$\textit{constraints } (2)\text{–}(3), \tag{11}$$

$$\textit{Benders cuts.} \tag{12}$$

Let $Q(x)$ denote the subproblem. For a given solution $\bar{x}$ from the master problem, the subproblem consists of objective (1) and constraints (4)–(9), which actually is an instance of the machine scheduling problem $1||\sum w_i C_i$, which can be solved by the WSPT rule (weighted shortest processing time first). Here, we use the cut by Laporte and Louveaux (1993):

$$\eta \geq (Q(\bar{x}) - L)(\sum_{\bar{x}_{ik} \in \bar{S}} x_{ik} - \sum_{\bar{x}_{ik} \notin \bar{S}} x_{ik} - |\bar{S}|) + Q(\bar{x}), \tag{13}$$

where $Q(\bar{x})$ is the optimal solution of the subproblem with given $\bar{x}$, $\bar{S}$ is the set of variables with values 1 in $\bar{x}$ and L is a lower bound for $Q(x)$ over x.

4 Computational results

We have obtained preliminary computational results for the methods described above. We have tested instances with $n = 16$, 20, and 24 and $m = 4$. Each family contains n/m jobs. The processing time of each family, job weights and sizes are generated from discrete uniform distributions on the intervals $[1, 100]$, $[1, 50]$ and $[1, 12]$, respectively. The machine capacity is 50. Each configuration has five instances. Each instance is solved to optimality by the model in Section 2 and the B&BC method. We use GUROBI as a linear solver. Some pre-processing procedures are implemented for both methods to reduce the solution spaces. In our preliminary tests, for each configuration, the B&BC routine requires less than half of the runtime of the model (1)-(9), on average.

References

Azizoglu M, Webster S (2001) Scheduling a batch processing machine with incompatible job families. Computers & Industrial Engineering 39(3-4):325–335

Dobson G, Nambimadom RS (1992) The batch loading and scheduling problem Research Report, Simon School of Business Administration, University of Rochester, Rochester, NY, 1992

Dobson G, Nambimadom RS (2001) The batch loading and scheduling problem. Operations Research 49(1):52–65

Laporte G, Louveaux FV (1993) The integer L-shaped method for stochastic integer programs with complete recourse. Operations Research Letters 13(3):133–142

Uzsoy R (1994) Scheduling a single batch processing machine with non-identical job sizes. International Journal of Production Research 32(7):1615–1635

MISTA 2019

Improved Algorithms for Scheduling with Job Delivery Coordination
(Extended Abstract)

Ling LIN · Yajing CHEN · Kaiwen TAN · Zhiyi TAN

1 Introduction

Supply chain management is one of the most important and active topics in operations research over the last twenty years [3]. Production and distribution are two key stages in a supply chain, and it is essential to consider the synergistic optimization of these two stages. One of the most typical models that reflect the idea is machine scheduling with job delivery coordination, which was first proposed by Lee and Chen [8].

Given a set $\mathcal{J}$ of jobs and a set $\mathcal{M}$ of m machines. Each job J_j has a processing time p_j and a size $s_j(\leq 1)$. For any subset $\mathcal{J}' \subseteq \mathcal{J}$, let $P(\mathcal{J}')$ and $S(\mathcal{J}')$ be the total processing time and the total size of the jobs in $\mathcal{J}'$, respectively. Each job should firstly be processed on a machine, then be delivered to a common customer by a single vehicle. The vehicle has a capacity of 1, i.e., it can load a subset of jobs with total size no more than 1. Such a subset is called a *batch*. A round trip between the machine and the customer takes constant time T. We aim to find a schedule of the jobs, as well as a feasible batch and delivery policy, such that the makespan is minimized.

There is a long line of studies on approximation algorithms for scheduling with job delivery coordination. For the single machine case, Chang and Lee [1] provided an algorithm with worst-case ratio $\frac{5}{3}$. Zhong et al. [10] presented a nearly best possible algorithm with worst-case ratio $\frac{3}{2}+\epsilon$. For the two machines case, the first algorithm was given by Chang and Lee [1], which has a worst-case ratio of 2. Recently, Chen et al. [2] presented a nearly best possible algorithm with worst-case ratio $\frac{3}{2}+\epsilon$. For the m machines case, the only work, to the authors knowledge, is an algorithm with worst-case ratio $\frac{7}{3}-\frac{1}{m}$ presented by Dong et al. [4].

In this paper, we present new algorithms for scheduling with job delivery coordination. An approximation algorithm with smaller worst-case ratio than that in [4] on

Ling LIN
School of Computer & Computing Science, Zhejiang University City College, Hangzhou 310015, PR China. Supported by the National Natural Science Foundation of China (11801505) and the Zhejiang Province Department of Education Research Projects (Y201533189).
E-mail: linl@zucc.edu.cn

Yajing CHEN · Kaiwen TAN · Zhiyi TAN
School of Mathematical Science, Zhejiang University, Hangzhou 310027, P. R. China. Supported by the National Natural Science Foundation of China (11671356).
E-mail: tanzy@zju.edu.cn

a small number of machines is given. We also design several heuristics for problems on a general number of machines and conduct numerical experiments, all these parts are omitted due to lack of place.

2 Bin-packing Algorithm DFFD

Scheduling and delivery problem with nonidentical job sizes have a close relationship with the one-dimensional bin-packing problem, where a sequence of jobs with sizes between 0 and 1 are required to be packed into a minimum number of unit capacity bins. *First Fit* (FF) [6] and *First Fit Decreasing* (FFD) [5] are two of the most fundamental algorithms for one-dimensional bin-packing. In this paper, we propose a new bin-packing algorithm *Delayed First Fit Deceasing* ($DFFD$ for short), which is a key point of our improved algorithm and may have its own uses in bin-packing problems. Additional to the job set $\mathcal{J}$, $DFFD$ needs as input a subset of jobs $\mathcal{J}_0 \subseteq \mathcal{J}$ with $S(\mathcal{J}_0) \leq 1$.

Algorithm $DFFD$

1. Pack jobs of $\mathcal{J}_0$ into the first bin $B_{[1]}$.

2. Sort jobs in $\mathcal{J} \setminus \mathcal{J}_0$ in non-increasing order by their sizes.

3. Packs the next unpacked job into the first opened bin that has enough room to accommodate it (including $B_{[1]}$). If no opened bin is suitable for this, a new bin is opened and the job is packed there.

We will use b^D to denote the number of bins employed when algorithm $DFFD$ is applied, and b^*_{BP} to denote the number of bins employed for a packing which uses a minimal number of bins. Note that $DFFD$ is also a kind of FF, and resembles FFD for majority of jobs. Though $DFFD$ is not superior to FFD in the worst-case ratio, the flexibility of $\mathcal{J}_0$ enables to obtain batches with special property. Meanwhile, $DFFD$ has well performance when b^*_{BP} is small. The proof of the following theorem is non-trivial, and is omitted due to lack of place.

Theorem 1 *If* $2 \leq b^*_{BP} \leq 4$, *then* $b^D \leq b^*_{BP} + 1$. *If* $5 \leq b^*_{BP} \leq 6$, *then* $b^D \leq b^*_{BP} + 2$.

3 Algorithms for Scheduling and Delivery

Algorithms for scheduling problems with job delivery coordination consists of three stages. The *batching* stage specifies how jobs are packed into different batches, which is essentially a one-dimensional bin-packing problem. The *scheduling* stage specifies how jobs are processed on machines, which is similar as parallel machine scheduling problem. The *delivery* stage specifies how batches are delivered to the customer.

Now we present procedures of different stages which will be called by the approximation algorithm AS. Procedures that only be used in the heuristics are omitted.

Procedure B1 for Batching Stage ([1])

1. Apply the FFD to $\mathcal{J}$.

Procedure B2 for Batching Stage

1. Construct an instance I_{KP} of the Knapsack problem with job set $\mathcal{J}$ and capacity 1. Apply an FPTAS of the Knapsack problem [7] to the I_{KP} to produce a subset $\mathcal{J}_0 \subseteq \mathcal{J}$ with $S(\mathcal{J}_0) \leq 1$ and $P(\mathcal{J}_0) \geq (1-\epsilon)P^*_{KP}$, where P^*_{KP} is the optimum of I_{KP}.

2. Apply the $DFFD$ to $\mathcal{J}$ with subset $\mathcal{J}_0$.

For any $i = 1, 2$, denote by b_i the number of batches opened by the Procedure Bi, and $B^i_{[l]}$ the lth opened batch, $l = 1, \ldots, b_i$.

Procedure S1 for Scheduling Stage ([9])

1. Reorder the batches $B^i_{[1]}, \ldots, B^i_{[b_i]}$ obtained in the Batching stage in non-decreasing order by total processing time. Let the lth batch of the new list be B^i_l.

2. Assign the batches from B^i_1 to $B^i_{b_i-1}$ one by one. Each batch is assigned to the machine that has the smallest load before it is assigned. All jobs in the same batch are assigned to the same machine as a whole.

3. Assign jobs in the last batch $B^i_{b_i}$ one by one. Each job in $B^i_{b_i}$ is assigned to the machine that has the smallest load before it is assigned.

Procedure D1 for Delivery Stage

1. Dispatch each finished but undelivered batch among $B^i_1, B^i_2, \ldots, B^i_{b_i-1}$ whenever the vehicle becomes available. If multiple batches among these batches have been completed when the vehicle becomes available, dispatch the batch with the earliest completion time.

2. Dispatch $B^i_{b_i}$ whenever it is finished and the vehicle becomes available.

Selecting one procedure from each stage leads to an algorithm which takes $\mathcal{J}$ as input and produces a schedule as output. For example, $(B2, S1, D1)$ represents the algorithm which runs procedures $B2, S1, D1$ successively. It is also allowed to run several such algorithms in parallel, and select the best schedule ever obtained.

Algorithm *AS*

1. Run Procedure $B1$.

2. If $b_1 = 1$ or $b_1 \geq 6$, run $(B1, S1, D1)$.

3. If $2 \leq b_1 \leq 5$, run $(B1, S1, D1)$ and $(B2, S1, D1)$, and select the better schedule.

Theorem 2 *The worst-case ratio of AS is* $\frac{5}{3} + \epsilon$ *when* $m = 3$*, where* $\epsilon > 0$ *can be arbitrary close to* 0*, and* 2 *when* $m = 4$.

References

1. Y. C. Chang, C. Y. Lee. Machine scheduling with job delivery coordination. *European Journal of Operational Research*, 158, 470-487 (2004).
2. Y. Chen Y, A. Zhang, Z. Y. Tan, Y. Xue, G. T. Chen. A ($\frac{3}{2} + \epsilon$)-approximation algorithm for scheduling on two parallel machines with job delivery coordination, *Journal of the Operational Research Society*, Doi: 10.1080/01605682.2019.1655204 (2019).
3. Z. L. Chen. Integrated production and outbound distribution scheduling: reviews and extensions. *Operations Research*, 58, 130-148 (2010).
4. J. Dong, X. Wang, L. Wang, J. Hu. Parallel matchine scheduling with job delivery coordination. *The ANZIAM Journal*, 58, 306-313 (2017).
5. G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is $FFD(L) \leq \frac{11}{9}OPT(L) + \frac{6}{9}$. *Lecture Notes in Computer Science*, 4614, 1-11 (2007).
6. G. Dósa, J. Sgall. First fit bin packing: a tight analysis. *Proceedings of the 30th Symposium on Theoretical Aspects of Computer Science*, 538-549 (2013).
7. O. H. Ibarra, C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22, 363-468 (1975).
8. C. Y. Lee, Z.-L. Chen. Machine scheduling with transportation considerations. *Journal of Scheduling*, 4, 3-24 (2001).
9. L. Y. Wang, Z. H. Liu. An improved algorithm for scheduling two identical machines with batch delivery consideration. *Operations Research Transactions*, 17(1), 38-43 (2013). (in Chinese)
10. W. Y. Zhong, G. Dosa, Z. Y. Tan. On the machine scheduling problem with job delivery coordination. *European Journal of Operational Research*, 182, 1057-1072 (2007).

MISTA 2019

Dynamic Incentive Schemes for Managing Dockless Bike-Sharing Systems

Huan Jin · Shaoxuan Liu · Rick So

Abstract Balancing supply and demand for a dockless bike-sharing system is difficult as users are allowed to pick up and return bikes anywhere within a large service area. In this paper we study the use of dynamic incentive schemes that offer rewards for users to pick up or return bikes outside a designated service zone based on the existing amount of bikes in the system. Our analysis shows that the firm should offer a higher reward for picking up or returning bikes outside the designated service zone when there exists a large number of bikes outside these areas. More importantly, we show that it is sufficient to offer only a few reward levels in order to achieve near-optimal performance. Using an extensive numerical experiments, we further illustrate specific operation environments under which these dynamic incentive schemes would be most effective in reducing the operating costs of the system.

1 Introduction

Bike-sharing systems provide an inexpensive, healthy and eco-friendly mode of local transportation, which have gained popularity around the world. Recently, dockless bike-sharing systems have received much attention, as they provide an added convenience for users who are allowed to pick up and return a shared bike anywhere within the service area. This additional convenience in a dockless system further helps to attract more users to utilize shared bikes, as it eliminates the need of users to walk to their closest docking stations which can be far from their pickup locations or final destinations.

While dockless bike-sharing systems are attractive to users, they also pose new operating challenges for the bike-sharing firms, as it could be costly to manage a

Huan Jin
School of Computer Science, University of Nottingham Ningbo China, Ningbo, Zhejiang,315100
E-mail: huan.jin@nottingham.edu.cn

Shaoxuan Liu
Antai College of Economics and Management, Shanghai Jiao Tong University, Shanghai, China
E-mail: liusx@sjtu.edu.cn

Rick So *
The Paul Merage School of Business, University of California, Irvine, California 92697, USA.
E-mail: rso@uci.edu

large amount of dockless bikes scattered in a large geographical service area. First, it is difficult to balance the supply and demand within the service area, which would result in low utilization or shortage of shared bikes in the system. Second, it increases the logistic costs for the bike-sharing firms to pick up these dockless bikes in such a large geographical area for maintenance/repair or inventory re-balancing purposes. Furthermore, dockless bikes left unattended in remote areas are more likely subject to theft or vandalism; see e.g. Pidd (2018). For public safety reasons, city ordinances could also impose strict limits or hefty fines for leaving a large amount of dockless bikes on crowded city streets, which can cause congestion or pedestrian hazards. Consequently, it is desirable for bike-sharing firms to restrict the amount of dockless bikes outside certain designated service areas where user demands are high, theft or vandalism is low, and these designated service areas are sufficiently compact to allow for efficient collections for maintenance or re-balancing.

Some dockless bike-sharing firms have recently explored the use of incentive schemes to help reduce the amount of their dockless bikes outside certain designated service areas. For example, Mobike (one of the largest dockless bike-sharing firms in China) has imposed an additional charge for users who return their bikes outside the designated service zone so as to encourage its users to return the bikes within these designated service areas that have high demand for better utilization as well as to facilitate an efficient way of picking up the bikes for maintenance or re-balancing purposes. In addition, Mobike has previously offered rewards for users who are willing to pick up any available bikes that are located outside these designated service areas. However, they do not find the use of such pickup rewards effective, as it could create the behavior of some users leaving the bikes outside the designated service area with the intention of picking up these bikes later to earn the pickup rewards.

Motivated by the use of the aforementioned incentive schemes to tackle the operating challenges in managing a dockless bike-sharing system, we develop an analytical framework to evaluate the effectiveness of deploying dynamic incentive schemes for managing the amount of dockless bikes in the system. Specifically, we allow the bike-sharing firm to offer a pickup reward for users who are willing to walk to pick up an available bike left outside the designated service area as well as to impose a penalty for users who return a bike outside the designated service area. To simplify our exposition, We consider a reward of returning a bike within the designated service area instead of a penalty of returning a bike outside the designated service area. Our modeling framework further allows the firm to "dynamically" adjust the amount of these pickup and return rewards depending on the existing number of dockless bikes outside the designated service area.

Our research aims to provide useful insights for a number of important research questions of practical significance. First, we analyze the behavior of the optimal dynamic incentive schemes and to determine how to set the optimal amount of incentives based on the existing number of dockless bikes outside the designated service area. Second, we evaluate the effectiveness of the use of these dynamic incentive schemes and provide specific operating conditions under which these dynamic incentive schemes would be most beneficial in reducing the operating costs of the system.

Our analysis shows that the firm should offer a higher amount in both the pickup reward and return reward when there exists a large number of bikes outside the designated service area. More importantly, we show that it is sufficient to offer only a few reward levels in order to achieve near-optimal performance. This result suggests that a

simple, yet practical, dynamic scheme can be used in managing a dockless bike-sharing system to minimize the operating costs of the system.

Using an extensive numerical experiments, we further illustrate specific operation environments under which the dynamic incentive schemes would be most effective in reducing the operating costs of the system. First, our results suggest that deploying pickup reward is always more effective than return reward except in situations when the demand rate for the designated service area is not significantly higher than the return rate outside the designated service area for which the pickup reward cannot induce a sufficient number of customers to pick up bikes outside the designated service area. Second, when the demand rate within the designated service area is higher than the return rate outside the designated service area, the benefit of using return reward to induce customers to return their bikes within the designated service area is very marginal, since the demand for bikes outside the designated service area combined with the pickup reward is sufficient to manage the number of bikes outside the designated service area. However, when the cost of leaving a bike outside the designated area is high, offering a return reward could still provide significant cost reduction in addition to offering a pickup reward. In this case, the pickup reward would serve as the primary incentive driver for increasing the number of customers to pick up bikes outside the designated service area, whereas the return reward can serve as a secondary incentive driver for reducing the number of bikes being returned outside the designated service area.

This paper is organized as follows. We provide a brief review of related literature in Section 2. In Section 3 we develop our modeling framework of using dynamic incentive schemes to manage shared bikes in a dockless system. In Section 4 we use a birth-and-death process to model the amount of dockless bikes in the system, and analyze the optimal dynamic incentive schemes in order to minimize the operating costs of the system. We further analyze the performance some special class of incentive schemes, and show that the optimal dynamic incentive scheme can be obtained as a limit of this special class of incentive schemes. In Section 5 we present the results of a comprehensive set of numerical experiments and illustrate several useful managerial insights based on our numerical results. We provide some concluding remarks in Section 6.

2 Literature Review

With increasing popularity of bike-sharing systems, there exists an extensive research literature on various operating issues involving bike-sharing. Bike-sharing systems with docking stations are especially well-studied, which includes location selections and sizing of the docking stations, redistribution and rebalancing of shared bikes at different docking stations, and optimization of truck routes in re-balancing bikes in the networks. We refer to two recent book chapters in this research area, Chou et al. (2019) and Fruend et al. (2019), which provide a comprehensive review of recent research papers that address these issues.

Recently, a number of scholars have studied various operational issues regarding dockless bike-sharing systems. For example, Caggiani et al. (2017) studied the use of a congestion pricing policy to plan and design equitable dockless bike-sharing systems. Pal and Zhang (2017) studied some rebalancing problems in dockless bike-sharing systems. Caggiani et al. (2018) proposed a modeling framework to dynamically manage dockless bikes-sharing systems. Cheng and Gao (2018) studied the optimal subscription

pricing strategy for dockless bike-sharing platforms adopted the monthly strategy in order to maximize the platform revenue. Chang et al. (2018) studied the faulty bike-sharing recycling problem in dockless bike-sharing systems. Zhou et al. (2019) studied the impact of dockless bike-sharing systems on housing prices.

Our paper is specifically concerned with related to the use of pricing mechanism to balance supply and demand in a dockless bike-sharing system. Recently, several researchers have studied the use of pricing mechanisms to match supply and demand in bike-sharing or vehicle-sharing systems. For vehicle sharing systems, Febbarao et al. (2012) discussed the use of fare discounts to incentivize customers to choose their destinations close to the zone where there is vehicle shortage. Waserhole et al. (2013) and Waserhole and Jost (2016) developed closed queueing network models to determine the optimal dynamic pricing policies to self regulate and improve system performance in vehicle sharing systems. Banerjee et al. (2017) also demonstrated the use of steady-state Markovian models for designing pricing policies in vehicle-sharing systems with multiple stations, and developed approximation schemes to solve the optimal pricing decisions. Jorge et al. (2015) studied the use of trip pricing strategy in vehicle sharing systems to induce a more balanced system in order to increase profitability of a vehicle-sharing system, and applies their results in a case study to demonstrate that the trip pricing strategy can be used to increase profit through a more balanced system.

For bike-sharing systems with docking stations, Pfrommer et al. (2014) studied the use of a dynamic incentive scheme to induce customers to change their target station, together with a tailored routing algorithm, to improve balancing in the system. Singla et al. (2015) proposed the use of monetary incentive schemes to incentivize users to improve system balance by providing them with alternate choices to pick or return bikes in a bike-sharing system with stations. They further evaluated the proposed incentive mechanisms through simulations based on data collected via a survey study and deployed the system in a real-world bike sharing system. Fricker and Gast (2016) also demonstrated the benefits of offering simple incentives to suggest users to return the bikes to less congested stations to improve system performance in a bike-sharing systems with stations. Haider et al. (2018) also studied the optimal price levels to incentivize users to pick up or return bikes at stations so as to strategically minimize the number of imbalanced stations in order to reduce the need for inventory re-positioning.

In contrast to all the above studies, we develop dynamic incentive schemes for managing supply and demand in a bike-sharing system with dockless stations. In dockless systems, the bikes can be scattered in any location within a large service zone, which make re-balancing by deploying an efficient truck collection strategy more challenging. As a result, the use of the incentive schemes to achieve balance could be an effective strategy. To the best of our knowledge, we are the first paper to study incentive mechanisms in balancing supply and demand in dockless bike-sharing systems.

3 Modeling Framework

For a tractable analysis, we make the following simplifying assumptions in our modeling framework:

1. We model the entire service zone as a circle of radius s, with the designated service area simply being as the center of the circle. There always exists shared bikes at the center of the circle to meet customer demand at the center, i.e., within the designated service area.

2. When there are n dockless bikes inside the circle (i.e., outside the designated service area), they are assumed to be randomly distributed in the circle at locations.
3. Users are heterogeneous with respect to their unit traveling cost c, and c is assumed to be uniformly distributed between $[0, 1]$.
4. All users have the real-time information (say, via a mobile app) for all available dockless bikes outside the center as well as the pickup and return rewards offered by the firm.

3.1 Consumer Choice Model

We next develop a consumer choice model to analyze how a user would choose to pick up as well as to return a shared bike either at the center (i.e., within the designated service area) or at her final destination inside the circle (i.e., outside the designated service area).

3.1.1 Bike Pickup

We consider two types of user demand: one occurs at the center (i.e., within the designated service area) and the other occurs inside the circle (i.e., outside the designated service area). We denote the arrival rate of these two demand types by d_0 and d_1, respectively.

For demand occurring at the center, users has planned to pick up the bike at the center. Suppose that the bike-sharing firm offers a pickup reward rate of p_n, with $0 \leq p(n) \leq 1$ (i.e., a pickup reward of $p(n)t$ for a dockless bike at a location of travel distance t, $0 < t \leq s$), if a user is willing to pick up a dockless bike at a location inside the circle rather than at the center and there are n available dockless bikes inside the circle. We assume that a user would be willing to pick up a dockless bike at any location only when his traveling cost ct to that location is less than the pickup reward $p(n)t$, or equivalently, $c < p(n)$. Since the unit traveling cost is assumed to be uniformly distributed on $[0, 1]$, the probability of an arriving user at the center who would pick up a dockless bike inside the circle with a reward rate of $p(n)$ is equal to $p(n)$.

For tractability, we assume that one of the available dockless bikes will be randomly selected when a user decides to pick up a dockless bike, i.e., $c < p(n)$.[1] Since the dockless bikes are assumed to be randomly distributed inside the circle, the probability that a (random) dockless bike is within travel distance of t from the center is equal to $\frac{\pi t^2}{\pi s^2} = \frac{t^2}{s^2}$. Therefore, the density function for any dockless bike with a travel distance of t from the center is given by $\frac{2t}{s^2}$. Then, the average travel distance to the dockless bike is equal to

$$\int_0^s t(\frac{2t}{s^2})dt = \frac{2s}{3}, \tag{1}$$

and the expected pickup reward paid to a user is equal to $\frac{2sp(n)}{3}$.

[1] It is also plausible to assume that the user would always choose to pick up the dockless bike at location of the largest travel distance, i.e., $T = \max\{t_1, t_2, ..., t_n\}$ where t_i denotes the travel distance between location L_i and the center. However, under this assumption, the locations of the dockless bikes would no longer be randomly distributed inside the circle since the farthest bikes from the center would always be the first ones to be picked up. Consequently, most dockless bikes would lie close to the center, and the distribution of $\{t_1, ..., t_n\}$ and $E(T)$ would be difficult to analyze under this assumption.

For demand occurring inside the circle, we assume that a user will always pick up the nearest dockless bike, including those at the center. If there is no available dockless bike inside the circle, we assume that the user will then go to the center to pick one up such that a demand of this type will always be met.

3.1.2 Bike Return

We denote the user rate of returning a bike inside the circle by d_2. We assume that a user can always choose to return the bike either at the center or simply at her final destination inside the circle. Suppose that the bike-sharing firm offers a return reward of $r(n) \geq 0$ if a user chooses to return the bike at the center instead of her final destination inside the circle when there are n available dockless bikes inside the circle. (For easier exposition here, we use a return reward if a user returns the bike to the center rather than incur a return penalty if a user returns the bike inside the circle.) Furthermore, the return reward is assumed to be the same, regardless of the exact return location.[2]

Consider a user of unit traveling cost of c with a return location of travel distance t from the center, where $0 < t \leq s$. The traveling cost for this user to return the bike at the center is then equal to ct, whereas the user can receive the return reward of $r(n)$.[3] Therefore, this user would choose to return the bike at the center only if $ct \leq r(n)$. Without loss of generality, we can assume that $r(n) \leq s$, as $c \leq 1$ and $t \leq s$ so that all users would opt to return their bikes at the center when $r(n) = s$, and there is no reason for the firm to offer a return reward greater than s.

Assume that the return location is randomly distributed inside the circle. Then, the probability that a user with return location of travel distance t from the center would choose to return the bike at the center is given by

$$\begin{aligned}
\beta_n &= \int_0^1 \left\{ \int_0^s 1_{[t < \frac{r(n)}{c}]} \frac{2t}{s^2} dt \right\} dc \\
&= \int_0^{\frac{r(n)}{s}} \left\{ \int_0^s \frac{2t}{s^2} dt \right\} dc + \int_{\frac{r(n)}{s}}^1 \left\{ \int_0^{\frac{r(n)}{c}} \frac{2t}{s^2} dt \right\} dc \\
&= \frac{r(n)}{s} + \frac{r(n)^2}{s^2} \int_{\frac{r(n)}{s}}^1 \frac{1}{c^2} dc \\
&= \frac{r(n)}{s} \left(2 - \frac{r(n)}{s} \right). \qquad (2)
\end{aligned}$$

3.2 Bike-Sharing Firm's Decision Problem

Suppose that the bike-sharing firm incurs an operational expense of h per unit time for maintaining each dockless bike inside the circle. We use the notation $(\mathbf{p}, \mathbf{r}) =$

[2] One alternate model is to use return reward rate in which the return reward is proportional to the travel distance from the return location to the center. However, the return location is private information of the user, and the firm would realize the location only after the user returns the bike. Thus, using a return award rate is difficult to implement in practice.

[3] Here we assume that the user does not need to travel back to location t after returning the bike at the center. If so, we can simply make the travel distance equal to $2t$ and re-scale the reward $r(n)$ accordingly.

$((p(1), p(2), ...), (r(1), r(2), ..))$ to denote the (dynamic) incentive scheme for the pickup and return rewards as described above. For any given dynamic incentive scheme $(\mathbf{p}, \mathbf{r})$, let π_n denote steady-state probability (assuming they exist) that there are n dockless bikes inside the circle. Then, the expected operating cost per unit time for the bike-sharing firm is given by

$$\Pi(\mathbf{p}, \mathbf{r}) = \sum_{n=1}^{\infty} \frac{2sp(n)}{3} d_0 p(n) \pi_n + \sum_{n=0}^{\infty} r(n) d_2 \beta_n \pi_n + \sum_{n=1}^{\infty} hn\pi_n, \tag{3}$$

where the first term represents the total expected pickup rewards paid to the users, the second term represents the total expected return rewards paid to the users, and the last term represents the additional operational expenses for maintaining all dockless bikes inside the circle.

The objective of the bike-sharing firm is to choose the optimal dynamic incentive scheme $(\mathbf{p}, \mathbf{r})$ in order to minimize the expected operating cost function given in (3). To solve this decision problem, we need to analyze the behavior of the number of dockless bikes inside the circle for any given dynamic incentive scheme. In the next section, we use a birth-and-death process to model the number of dockless bikes inside the circle, from which we can analyze the behavior of the optimal dynamic incentive scheme for the firm's decision problem.

4 Model Analysis

We model the number of dockless bikes inside the circle as a birth-and-death process. Specifically, the return of a bike increases the number of dockless bikes inside the circle by one if the user chooses to return the bike at her final destination. This corresponds to a "birth" in the process, which increases the number of units inside the circle from n to $(n+1)$ with a transition rate of

$$\lambda_n = d_2(1 - \beta_n), \tag{4}$$

where β_n is given in (2). Similarly, a bike pickup decreases the number of dockless bikes inside the circle by one if the user chooses to pick up an available dockless bike inside the circle. This corresponds to a "death" in the process, which decreases the number of units inside the circle from n to $(n-1)$ (for $n > 0$) with a transition rate of

$$\mu_n = d_0 p(n) + d_1. \tag{5}$$

We can follow the standard analysis for a birth-and-death process to find the steady-state probability of having n units in the system, denoted by π_n, with a birth rate of λ_n and a death rate of μ_n given by (4) and (5), respectively. Specifically, define $x_0 = 1$ and

$$x_n = \frac{\lambda_{n-1}}{\mu_n} x_{n-1} = \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i}, \quad n \geq 1. \tag{6}$$

We consider only admissible incentive schemes such that $\sum_{n=0}^{\infty} x_n < \infty$. For any admissible incentive scheme, the steady-state probabilities π_n is given by

$$\pi_0 = \left(\sum_{n=0}^{\infty} x_n \right)^{-1}$$

$$\pi_n = \pi_0 x_n \qquad n \geq 1. \tag{7}$$

Then, we can use (3) to compute the total expected cost per unit time associated with each admissible incentive scheme.

To determine the optimal dynamic incentive scheme, one needs to evaluate all admissible incentive scheme to find the one with the minimum total expected cost. As there are infinite ways of setting the pickup and return rewards $(\mathbf{p}, \mathbf{r})$, we first evaluate some simple special classes of dynamic incentive schemes, and then show that the optimal incentive scheme can be obtained as a limit of these special classes of dynamic incentive schemes.

4.1 One-tier Incentive Schemes

We first consider the simplest incentive scheme where $p(n) = p$ and $r(n) = r$ for all $n \geq 0$. In other words, the firm offers a fixed pickup reward rate and a fixed return reward regardless of the number of dockless bikes inside the circle. Thus, this class of incentive schemes is static and independent of the number of dockless bikes inside the circle, which also includes the special case in which the firm does not offer any pickup or return reward, i.e., $p(n) = r(n) = 0$ for all n.

In this case, the birth rate and death rate of the system are simply given by $\lambda_n = d_2\left[1 - \frac{r}{s}\left(2 - \frac{r}{s}\right)\right]$ and $\mu_n = d_0 p + d_1$, respectively. Define

$$\rho = \frac{d_2\left[1 - \frac{r}{s}\left(2 - \frac{r}{s}\right)\right]}{d_0 p + d_1}. \tag{8}$$

Proposition 1 *Consider any (p, r) such that $\rho < 1$. Then, the expected cost function (3) is given by*

$$\Pi_1(p, r) = \frac{2d_0 s p^2 \rho}{3} + \frac{d_2 r^2}{s}\left(2 - \frac{r}{s}\right) + \frac{h\rho}{1-\rho}. \tag{9}$$

Proof of Proposition 1: In this case, the system corresponds to the classical $M/M/1$ queueing system, and the steady-state probabilities are given by $\pi_n = (1-\rho)\rho^n$, for $n \geq 0$. Then, the expected cost function (3) can be written as

$$\begin{aligned}\Pi_1(p, r) &= \sum_{n=1}^{\infty} \frac{2sp}{3}(d_0 p)(1-\rho)\rho^n + \sum_{n=0}^{\infty} r d_2 \frac{r}{s}\left(2 - \frac{r}{s}\right)(1-\rho)\rho^n + \sum_{n=1}^{\infty} hn(1-\rho)\rho^n \\ &= \frac{2d_0 s p^2 \rho}{3} + \frac{d_2 r^2}{s}\left(2 - \frac{r}{s}\right) + \frac{h\rho}{1-\rho}.\end{aligned}$$

□

For the class of admissible 1-tier incentive schemes that satisfy $\rho < 1$, we can determine the optimal values of p and r by solving the following optimization problem with two variables:

$$\begin{aligned} Minimize \quad & \Pi_1(p, r) \\ Subjectto \quad & 0 \leq p \leq 1, \\ & 0 \leq r \leq s, \\ & 0 \leq \rho < 1. \end{aligned} \tag{10}$$

4.2 Two-tier Incentive Schemes

We next extend our analysis of one-tier incentive schemes to two-tier incentive schemes. In a two-tier incentive scheme, the firm offers two incentive levels based on whether the number of dockless bikes exceeds a certain threshold. In particular, the structure of a two-tier incentive scheme can be expressed as follows:

$$p(n) = \begin{cases} p_1, \, if \; 0 < n \leq N_1 \\ p_2, \, if \; n > N_1 \end{cases} \tag{11}$$

$$r(n) = \begin{cases} r_1, \, if \; 0 \leq n < N_1 \\ r_2, \, if \; n \geq N_1, \end{cases} \tag{12}$$

with $0 \leq p_1, p_2, r_1, r_2 \leq 1$ and $N_1 \geq 1$. Note that a two-tier incentive scheme corresponds to a one-tier incentive scheme when $p_1 = p_2$ and $r_1 = r_2$.

Define

$$\rho_1 = \frac{d_2 \left[1 - \frac{r_1}{s}\left(2 - \frac{r_1}{s}\right)\right]}{d_0 p_1 + d_1} \tag{13}$$

and

$$\rho_2 = \frac{d_2 \left[1 - \frac{r_2}{s}\left(2 - \frac{r_2}{s}\right)\right]}{d_0 p_2 + d_1}. \tag{14}$$

Proposition 2 *Consider any (p_2, r_2) such that $\rho_2 < 1$. Then, the expected cost function (3) is given by*

$$\begin{aligned} \Pi_2(p_1, p_2, r_1, r_2, N_1) = {} & \frac{2sd_0\pi_0}{3}\left\{\frac{p_1^2\rho_1(1-\rho_1^{N_1})}{1-\rho_1} + \frac{p_2^2\rho_1^{N_1}\rho_2}{1-\rho_2}\right\} \\ & + d_2\pi_0\left\{\frac{\frac{r_1^2}{s}\left(2-\frac{r_1}{s}\right)(1-\rho_1^{N_1})}{1-\rho_1} + \frac{\frac{r_2^2}{s}\left(2-\frac{r_2}{s}\right)\rho_1^{N_1}}{1-\rho_2}\right\} \\ & + h\pi_0\left\{\frac{\rho_1[1-(N_1+1)\rho_1^{N_1}+N_1\rho_1^{N_1+1}]}{(1-\rho_1)^2} + \frac{\rho_1^{N_1}\rho_2[N_1(1-\rho_2)+1]}{(1-\rho_2)^2}\right\}. \end{aligned} \tag{15}$$

Proof of Proposition 2: In this case, it follows from (4) and (5) that

$$\begin{aligned} \lambda_n &= \begin{cases} d_2\left[1 - \frac{r_1}{s}\left(2 - \frac{r_1}{s}\right)\right], \, if \; 0 \leq n < N_1 \\ d_2\left[1 - \frac{r_2}{s}\left(2 - \frac{r_2}{s}\right)\right], \, if \; n \geq N_1 \end{cases} \\ \mu_n &= \begin{cases} d_0 p_1 + d_1, & if \; 0 < n \leq N_1 \\ d_0 p_2 + d_1, & if \; n > N_1. \end{cases} \end{aligned} \tag{16}$$

Then, we substitute (16) into (6) and (7) to obtain $\mathrm{x}_n = \begin{cases} \rho_1^n, & for \; n \leq N_1 \\ \rho_1^{N_1}\rho_2^{n-N_1} & for \; n > N_1, \end{cases}$ and

$$\pi_0 = \left(\frac{1-\rho_1^{N_1+1}}{1-\rho_1} + \rho_1^{N_1}\frac{\rho_2}{1-\rho_2}\right)^{-1} \tag{17}$$

$$\pi_n = \begin{cases} \pi_0\rho_1^n, & for \; 0 \leq n \leq N_1 \\ \pi_0\rho_1^{N_1}\rho_2^{n-N_1} & for \; n > N_1. \end{cases} \tag{18}$$

We now substitute (17) and (18) into the expected cost function (3) and obtain

$$\begin{aligned}\Pi_2(p_1,p_2,r_1,r_2,N_1) &= \left\{\sum_{n=1}^{N_1}\frac{2sp_1}{3}d_0p_1\pi_n+\sum_{n=N_1+1}^{\infty}\frac{2sp_2}{3}d_0p_2\pi_n\right\}\\ &\quad+\left\{\sum_{n=0}^{N_1-1}\frac{d_2r_1^2}{s}\left(2-\frac{r_1}{s}\right)\pi_n+\sum_{n=N_1}^{\infty}\frac{d_2r_2^2}{s}\left(2-\frac{r_2}{s}\right)\pi_n\right\}+\sum_{n=1}^{\infty}hn\pi_n\\ &= \frac{2sd_0\pi_0}{3}\left\{\frac{p_1^2\rho_1(1-\rho_1^{N_1})}{1-\rho_1}+\frac{p_2^2\rho_1^{N_1}\rho_2}{1-\rho_2}\right\}\\ &\quad+d_2\pi_0\left\{\frac{\frac{r_1^2}{s}\left(2-\frac{r_1}{s}\right)(1-\rho_1^{N_1})}{1-\rho_1}+\frac{\frac{r_2^2}{s}\left(2-\frac{r_2}{s}\right)\rho_1^{N_1}}{1-\rho_2}\right\}\\ &\quad+h\pi_0\left\{\frac{\rho_1[1-(N_1+1)\rho_1^{N_1}+N_1\rho_1^{N_1+1}]}{(1-\rho_1)^2}+\frac{\rho_1^{N_1}\rho_2[N_1(1-\rho_2)+1]}{(1-\rho_2)^2}\right\}\end{aligned}$$

where the last two terms can be derived using the fact that

$$\begin{aligned}\sum_{n=1}^{N_1}n\rho_1^n &= \sum_{n=1}^{N_1}(n+1)\rho_1^n-\sum_{n=1}^{N_1}\rho_1^n=\sum_{n=1}^{N_1}\frac{\partial}{\partial\rho_1}(\rho_1^{n+1})-\frac{\rho_1(1-\rho_1^{N_1})}{1-\rho_1}\\ &= \frac{\partial}{\partial\rho_1}\left(\sum_{n=1}^{N_1}\rho_1^{n+1}\right)-\frac{\rho_1(1-\rho_1^{N_1})}{1-\rho_1}=\frac{\partial}{\partial\rho_1}\left(\frac{\rho_1^2(1-\rho_1^{N_1})}{1-\rho_1}\right)-\frac{\rho_1(1-\rho_1^{N_1})}{1-\rho_1}\\ &= \frac{(1-\rho_1)[2\rho_1-(N_1+2)\rho_1^{N_1+1}]+\rho_1^2(1-\rho_1^{N_1})}{(1-\rho_1)^2}-\frac{\rho_1(1-\rho_1^{N_1})}{1-\rho_1}\\ &= \frac{\rho_1[1-(N_1+1)\rho_1^{N_1}+N_1\rho_1^{N_1+1}]}{(1-\rho_1)^2}\end{aligned}$$

and

$$\begin{aligned}\sum_{n=N_1+1}^{\infty}n\rho_1^{N_1}\rho_2^{n-N_1} &= \rho_1^{N_1}\sum_{n=1}^{\infty}(N_1+n)\rho_2^n=\rho_1^{N_1}\left(N_1\sum_{n=1}^{\infty}\rho_2^n+\sum_{n=1}^{\infty}n\rho_2^n\right)\\ &= \rho_1^{N_1}\left(N_1\frac{\rho_2}{1-\rho_2}+\frac{\rho_2}{(1-\rho_2)^2}\right)\\ &= \frac{\rho_1^{N_1}\rho_2[N_1(1-\rho_2)+1]}{(1-\rho_2)^2}.\end{aligned}$$

□

For the class of admissible 2-tier incentive schemes that satisfy $\rho_2 < 1$, we can determine the optimal values of $(p_1, p_2, r_1, r_2, N_1)$ by solving the following optimization problem:

$$\begin{aligned}&Minimize && \Pi_2(p_1,p_2,r_1,r_2,N_1)\\ &Subjectto && 0\le p_1,p_2\le 1,\\ &&& 0\le r_1,r_2\le s\\ &&& N_1\ge 1,\quad N_1: integer\\ &&& 0\le\rho_2<1.\end{aligned}$$

4.3 Multi-tier Incentive Schemes

We now generalize our analysis to multi-tier incentive schemes. For a k-tier incentive scheme, the firm offers k incentive levels based on the number of dockless bikes inside the circle. In particular, the structure of a k-tier incentive scheme consists of $(k-1)$ thresholds $(N_1, N_2, ..., N_{k-1})$, with $N_0 = 0 < N_1 < N_2 < ... < N_{k-1} < N_k = \infty$, and the pickup reward rates $p(n)$ and return rewards $r(n)$ are defined as follows:

$$p(n) = p_i, \; for \; N_{i-1} < n \leq N_i \tag{19}$$
$$r(n) = r_i, \; for \; N_{i-1} \leq n < N_i \tag{20}$$

with $0 \leq p_1, ..., p_k, r_1, ..., r_k \leq 1$.

Define

$$\rho_i = \frac{d_2\left[1 - \frac{r_i}{s}\left(2 - \frac{r_i}{s}\right)\right]}{d_0 p_i + d_1}. \quad for \; i = 1, 2, ..., k. \tag{21}$$

Proposition 3 *Consider any (p_k, r_k) such that $\rho_k < 1$. Then, the expected cost function (3) is given by*

$$\begin{aligned}
\Pi_k(p_i, r_i, N_i) = {} & \frac{2sd_0\pi_0}{3}\left\{\sum_{i=1}^{k-1}\frac{p_i^2\left(\prod_{j=1}^{i-1}\rho_j^{N_j - N_{j-1}}\right)\rho_i(1-\rho_i^{N_i - N_{i-1}})}{1-\rho_i} + \frac{p_k^2\left(\prod_{j=1}^{k-1}\rho_j^{N_j-N_{j-1}}\right)\rho_k}{1-\rho_k}\right\} \\
& + d_2\pi_0\left\{\sum_{i=1}^{k-1}\frac{\frac{r_i^2}{s}\left(2-\frac{r_i}{s}\right)\left(\prod_{j=1}^{i-1}\rho_j^{N_j-N_{j-1}}\right)(1-\rho_i^{N_i-N_{i-1}})}{1-\rho_i} + \frac{\frac{r_k^2}{s}\left(2-\frac{r_k}{s}\right)\left(\prod_{j=1}^{k-1}\rho_j^{N_j-N_{j-1}}\right)}{1-\rho_k}\right\} \\
& + h\pi_0\left\{\sum_{i=1}^{k-1}\frac{\left(\prod_{j=1}^{i-1}\rho_j^{N_j-N_{j-1}}\right)\rho_i\left[N_{i-1}(1-\rho_i)+1-(N_i+1)\rho_i^{N_i-N_{i-1}}+N_i\rho_i^{N_i-N_{i-1}+1}\right]}{(1-\rho_i)^2}\right. \\
& \left. + \frac{\left(\prod_{j=1}^{k-1}\rho_j^{N_j-N_{j-1}}\right)\rho_k[N_{k-1}(1-\rho_k)+1]}{(1-\rho_k)^2}\right\}.
\end{aligned} \tag{22}$$

Proof of Proposition 3: In this case, it follows from (4) and (5) that

$$\begin{aligned}
\lambda_i &= d_2\left[1 - \frac{r_1}{s}\left(2 - \frac{r_1}{s}\right)\right] \qquad for \; N_{i-1} < n \leq N_i \\
\mu_i &= d_0 p_1 + d_1 \qquad for \; N_{i-1} < n \leq N_i.
\end{aligned} \tag{23}$$

Then, we substitute and (23) into (6) and (7) to deduce that $\mathrm{x}_n = \left\{\prod_{j=1}^{i-1}\rho_j^{N_j-N_{j-1}}\right\}\rho_i^{n-N_{i-1}}, \quad for \; N_{i-1} < n \leq N_i$, and that the steady-state probabilities are given by

$$\pi_0 = \left\{\frac{1-\rho_1^{N_1+1}}{1-\rho_1} + \sum_{i=1}^{k-2}\left(\frac{\rho_{i+1}(1-\rho_{i+1}^{N_{i+1}-N_i})}{1-\rho_{i+1}}\prod_{j=1}^{i}\rho_j^{N_j-N_{j-1}}\right) + \frac{\rho_k}{1-\rho_k}\prod_{i=1}^{k-1}\rho_i^{N_i-N_{i-1}}\right\}^{-1} \tag{24}$$

$$\pi_n = \pi_0\left\{\prod_{j=1}^{i-1}\rho_j^{N_j-N_{j-1}}\right\}\rho_i^{n-N_{i-1}}, \quad for \; N_{i-1} < n \leq N_i. \tag{25}$$

We now substitute (24) and (25) into the expected cost function (3) to obtain

$$\begin{aligned}
\Pi_k(p_i, r_i, N_i) &= \sum_{i=1}^{k} \sum_{n=N_{i-1}+1}^{N_i} \frac{2sp_i}{3} d_0 p_i \pi_n + \sum_{i=1}^{k} \sum_{n=N_{i-1}}^{N_i-1} \frac{d_2 r_i^2}{s} \left(2 - \frac{r_i}{s}\right) \pi_n + \sum_{n=1}^{\infty} hn\pi_n \\
&= \frac{2sd_0\pi_0}{3} \left\{ \sum_{i=1}^{k-1} \frac{p_i^2 \left(\prod_{j=1}^{i-1} \rho_j^{N_j - N_{j-1}}\right) \rho_i (1 - \rho_i^{N_i - N_{i-1}})}{1 - \rho_i} + \frac{p_k^2 \left(\prod_{j=1}^{k-1} \rho_j^{N_j - N_{j-1}}\right) \rho_k}{1 - \rho_k} \right\} \\
&\quad + d_2\pi_0 \left\{ \sum_{i=1}^{k-1} \frac{\frac{r_i^2}{s}\left(2 - \frac{r_i}{s}\right) \left(\prod_{j=1}^{i-1} \rho_j^{N_j - N_{j-1}}\right) (1 - \rho_i^{N_i - N_{i-1}})}{1 - \rho_i} + \frac{\frac{r_k^2}{s}\left(2 - \frac{r_k}{s}\right) \left(\prod_{j=1}^{k-1} \rho_j^{N_j - N_{j-1}}\right)}{1 - \rho_k} \right\} \\
&\quad + h\pi_0 \left\{ \sum_{i=1}^{k-1} \frac{\left(\prod_{j=1}^{i-1} \rho_j^{N_j - N_{j-1}}\right) \rho_i \left[N_{i-1}(1 - \rho_i) + 1 - (N_i + 1)\rho_i^{N_i - N_{i-1}} + N_i \rho_i^{N_i - N_{i-1}+1}\right]}{(1 - \rho_i)^2} \right. \\
&\qquad \left. + \frac{\left(\prod_{j=1}^{k-1} \rho_j^{N_j - N_{j-1}}\right) \rho_k [N_{k-1}(1 - \rho_k) + 1]}{(1 - \rho_k)^2} \right\}.
\end{aligned}$$

□

For the class of admissible k-tier incentive schemes that satisfy $\rho_k < 1$, We can determine the optimal values of $(p_1, ..., p_k, r_1, ..., r_k, N_1, ...N_{k-1})$ by solving the following optimization problem:

$$\begin{aligned}
&Minimize \quad \Pi_k(p_i, r_i, N_i) \qquad (26)\\
&Subjectto \quad 0 \le p_i \le 1 \quad for\ i = 1, 2, ..., k \\
&\qquad\quad 0 \le r_i \le s \quad for\ i = 1, 2, ..., k \\
&\qquad\quad 1 \le N_1 < N_2 < ...N_{k-1}, \quad N_1, ..., N_{k-1} : integers \\
&\qquad\quad 0 \le \rho_k < 1.
\end{aligned}$$

4.4 Optimal Dynamic Incentive Schemes

Let Π_k^* denote the minimum total expected cost among the class of k-tier incentive schemes, and Π^* denote the minimum total expected cost among all admissible policies. Observe that the class of k-tier incentive schemes include all $(k-1)$-tier incentive schemes, as we can simply set $p_{i-1} = p_i$ and $r_{i-1} = r_i$ for some $1 < i \le k$. Therefore, it is clear that $\Pi_{k-1}^* \ge \Pi_k^*$. Furthermore, as $k \to \infty$, the class of k-tier incentive schemes expands to include all admissible incentive schemes. Therefore, we have the following result, which shows that the optimal incentive scheme can be obtained as a limit of the optimal multi-tier incentive

Proposition 4 *$\Pi_{k-1}^* \ge \Pi_k^*$ for all $k \ge 2$, and $\lim_{k\to\infty} \Pi_k^* = \Pi^*$.*

4.5 Numerical Analysis

We conducted numerical analysis to study the structure of the optimal k-tier incentive schemes as well as the convergence of the optimal k-tier incentive schemes to the optimal dynamic scheme as shown in Proposition 4. From our numerical analysis, we observe several interesting insights that have practical implications as summarized below.

First, the optimal k-tier incentive schemes exhibit the following monotonic property: $p_1^* < p_2^* < ... < p_k^*$ and $r_1^* < r_2^* < ... < r_k^*$ for all values of k. In other words, the firm should offer both higher pickup and return rewards as the number of bikes outside the designated service area increases. This property is reasonable and intuitive. Table 1 shows the optimal $k-$tier incentive schemes for the numerical example with $s = 1$, $d_0 = 1000$, $d_1 = 100$, $d_2 = 500$ and $h = 10$, with $k = 1, 2, 3, 4, 5$. Note that the last two columns demonstrate the monotonic property as discussed above.

Table 1 Results for the optimal $k-$tier incentive schemes.

k	$\Pi_k^* = \Pi(\mathbf{p}^*, \mathbf{r}^*, \mathbf{N}^*)$	$\mathbf{N}^*$	$\mathbf{p}^*$	$\mathbf{r}^*$
1	118.51	-	.337	.211
2	111.23	2	(.229, .380)	(.146, .235)
3	109.97	(1, 3)	(.187, .289, .402)	(.120, .182, .248)
4	109.48	(1, 2, 4)	(.186, .263, .333, .423)	(.120, .167, .208, .259)
5	109.28	(1, 2, 3, 5)	(.186, .262, .314, .368, .444)	(.120, .167, .197, .229, .271)

Second, the value of Π_k^* decreases significantly as k increases for only small values of k, while the marginal cost reduction decreases rapidly as k increases, and usually becomes very minimal for $k \geq 5$. To illustrate this property for the numerical example given in Table 1, we further plot the minimum operating costs Π_k^* for higher values of k in Figure 1. In this example, the optimal 2-tier incentive scheme provides a substantial cost reduction over the optimal 1-tier incentive scheme, while the marginal cost reduction beyond 3-tier incentive scheme becomes very minimal. More importantly, it shows that the minimum cost under the optimal multi-tier incentive schemes converges to the overall optimal dynamic incentive scheme very rapidly as k increases. Our result thus implies that it is necessary to deploy only a very small number of tiers (i.e., a small value of k) to achieving near-optimal results. A small value of k makes the multi-tier incentive schemes much simpler to implement in practice.

Third, the optimal values of $\mathbf{N}^*$, $\mathbf{p}^*$ and $\mathbf{r}^*$ between two successive optimal incentive scheme exhibit the following interesting property. Suppose that the optimal k-tier incentive scheme is given by $(p_1^k, ..., p_k^k, r_1^k, ..., r_k^k, N_1^k, ..., N_{k-1}^k)$, and the optimal $(k+1)$-tier incentive scheme is given by $(p_1^{k+1}, ..., p_{k+1}^{k+1}, r_1^{k+1}, ..., r_{k+1}^{k+1}, N_1^{k+1}, ..., N_k^{k+1})$, respectively. Then,

$$\begin{aligned}
&0 \leq p_1^{k+1} \leq p_1^k, \quad p_{i-1}^k \leq p_i^{k+1} \leq p_i^k, \ i = 2, ..., k, \quad p_k^k \leq p_{k+1}^{k+1} \leq 1 \\
&0 \leq r_1^{k+1} \leq r_1^k, \quad r_{i-1}^k \leq r_i^{k+1} \leq r_i^k, \ i = 2, ..., k, \quad r_k^k \leq r_{k+1}^{k+1} \leq s \\
&1 \leq N_1^{k+1} \leq N_1^k, \quad N_{i-1}^k \leq N_i^{k+1} \leq N_i^k, \ i = 2, ..., k-1, \quad N_{k-1}^k \leq N_k^{k+1}.
\end{aligned} \tag{27}$$

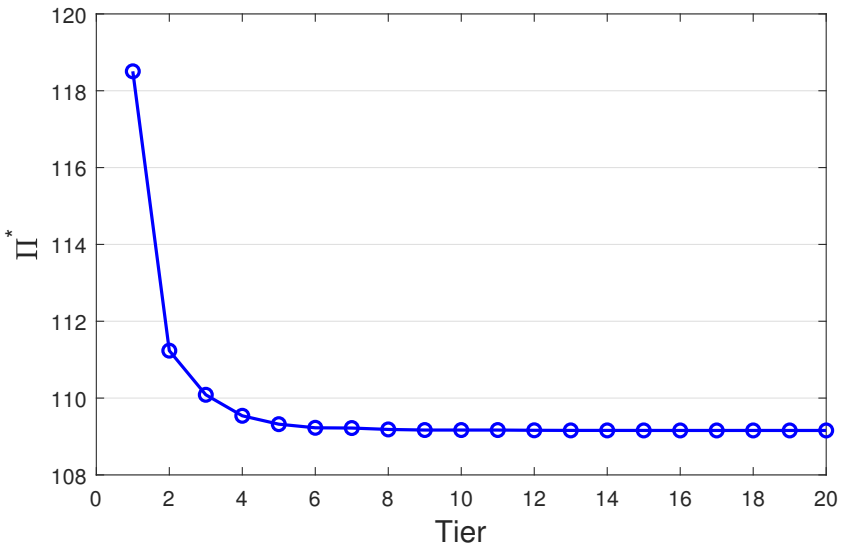


Fig. 1 Converge of objective values

This property demonstrates that a higher-tier scheme essentially refines a lower-tier scheme by adjusting the pickup and return rewards around the lower-tier results.

For example, observe from Table 1 that $p_1^* < p^* < p_2^*$ and $r_1^* < r^* < r_2^*$ when comparing the results between the optimal 1-tier and 2-tier incentive schemes. In particular, the firm would set the optimal pickup reward rate at $p^* = .337$ for the optimal 1-tier scheme. However, under the optimal 2-tier scheme, the firm would reduce the optimal pickup reward rate to $p_1^* = .229$ when there is less than 2 bikes outside the designated service area, but would increase the optimal pickup reward rate to $p_2^* = .380$ when there are 2 or more bikes outside the designated service area. Similar adjustments would also be made to the return rewards accordingly.

We can also exploit the last property given in (27) to design an efficient algorithm for finding the optimal k-tier incentive scheme in an iterative manner as follows. Specifically, suppose that the optimal k-tier incentive schemes is given by $(\mathbf{p}^*, \mathbf{r}^*, \mathbf{N}^*) = (p_1^*, ..., p_k^*, r_1^*, ..., r_k^*, N_1^*, ..., N_{k-1}^*)$. Then, we can restrict the search for the optimal $(k+1)$-tier incentive scheme by solving the following optimization problem:

$$
\begin{aligned}
&\textit{Minimize} \quad \Pi_{k+1}(p_i, r_i, N_i) \qquad (28)\\
&\textit{Subjectto} \quad 0 \le p_1 \le p_1^*, \quad p_{i-1}^* \le p_i \le p_i^*, \ i = 2, ..., k, \quad p_k^* \le p_{k+1} \le 1\\
&\qquad\qquad 0 \le r_1 \le r_1^*, \quad r_{i-1}^* \le r_i \le r_i^*, \ i = 2, ..., k, \quad r_k^* \le r_{k+1} \le s\\
&\qquad\qquad 1 \le N_1 \le N_1^*, \quad N_{i-1}^* \le N_i \le N_i^*, \ i = 2, ..., k-1, \quad N_{k-1}^* \le N_k,\\
&\qquad\qquad N_1, ..., N_k : integers\\
&\qquad\qquad 0 \le \rho_{k+1} < 1.
\end{aligned}
$$

The optimization problem (28) can be solved much more efficiently than the original optimization problem (26), especially for high values of k.

For our numerical experiments discussed in the next section, we first use (28) to solve for the optimal k-tier incentive scheme iteratively, starting with $k = 1$. We then determine the corresponding optimal dynamic incentive scheme by the convergence of the optimal k-tier schemes as stated in Proposition 4. In particular, we reach the optimal minimum expected cost when the percent difference in the minimum expected cost between two successive optimal k-tier incentive schemes is less than some tolerance amount, i.e., $\frac{\Pi_k^* - \Pi_{k+1}^*}{\Pi_k^*} < \epsilon$. We set $\epsilon = 0.0001$ in all our numerical experiments.

5 Effectiveness of Dynamic Incentive Schemes

We conducted a comprehensive set of numerical experiments to evaluate the effectiveness of using the dynamic incentive schemes in managing the number of dockless bikes outside the designated service area. Based on the results of our numerical experiments, we provide a number of managerial insights that would be useful in selecting the appropriate incentive schemes under different operating conditions.

Our general model deploys the use of both a dynamic pickup reward (p_i) and return reward (r_i) to incentivize customers to pick up bikes outside the designated service area and to return bikes inside the designated service area, respectively. We shall refer to this scheme as the dual reward scheme. As it requires additional efforts in managing either the pickup or return reward, it is natural to ask whether the use of only one such reward would be sufficient to achieve near-optimal results. Therefore, we also consider two simpler incentive mechanisms in which the firm would deploy only one type of reward. These corresponds to two special cases of our general model with $r_i = 0$ or $p_i = 0$ for all i. We shall refer to these two special schemes as the pickup reward scheme and return reward scheme, respectively.

We next provide a set of numerical results to illustrate a number of main insights from our numerical experiments. In this set of numerical experiments, we set the value of $d_2 = 100$, which represents the return rate of bikes outside the designated service area, but varied the values of d_0, d_1, h and s to assess the potential benefits of various dynamic incentive schemes.

Table 2 summarizes the results for the cases with $s = 1$ and $h = 0.1$ and different values of d_0 and d_1. In each case, we provide the minimum total expected cost under the optimal dual reward scheme, the optimal pickup reward scheme and the optimal return reward scheme. For comparison purposes, we also include the minimum total expected cost when the firm does not offer any pickup or return rewards, which is referred to as the no reward scheme. Note that under the no reward scheme, the minimize total expected cost is finite only if the pickup rate is higher than the return rate of bikes outside the designated service area, i.e., $d_1 > d_2 = 100$.

First, observe from Table 2 that the pickup reward scheme always outperforms the return reward scheme except for the cases when $d_0 = 100$. This suggests that using pickup reward is always more effective than using return reward except when the demand rate for the designated service area is not significantly higher than the return rate outside the designated service area for which the pickup reward is not effective in inducing sufficient customers to travel to pick up the bikes outside the designated service area. Instead, return reward is more effective to incentivize customers to return their bikes within the designated service area. For all other cases with d_0 =500, 1000, and 2000, the optimal pickup reward scheme provides a much lower total expected cost than that of the optimal return reward scheme.

Table 2 Minimum cost under different incentive schemes with $s = 1$, $d_2 = 100$, and $h = 0.1$.

d_0	d_1	Dual Reward	Pickup Reward ($r_i = 0$)	Return Reward ($p_i = 0$)	No Reward ($r_i = p_i = 0$)
100	50	9.23	19.44	16.73	∞
	75	3.53	6.70	5.41	∞
	100	1.03	1.41	1.23	∞
	125	0.36	0.40	0.37	0.40
	150	0.20	0.20	0.20	0.20
500	50	4.05	4.84	16.73	∞
	75	1.80	2.15	5.41	∞
	100	0.74	0.82	1.23	∞
	125	0.33	0.34	0.37	0.40
	150	0.19	0.20	0.20	0.20
1000	50	2.52	2.77	16.73	∞
	75	1.28	1.38	5.41	∞
	100	0.61	0.64	1.23	∞
	125	0.31	0.32	0.37	0.40
	150	0.19	0.19	0.20	0.20
2000	50	1.59	1.68	16.73	∞
	75	0.90	0.93	5.41	∞
	100	0.49	0.50	1.23	∞
	125	0.28	0.28	0.37	0.40
	150	0.18	0.18	0.20	0.20

Furthermore, when both d_1 and d_0 is greater than $d_2 = 100$, the minimum total expected cost of the optimal pickup reward scheme is very close to that of the optimal dual reward scheme. This suggests that the use of pickup reward alone can achieve near-optimal results, and the additional benefit of offering return reward is very small. In other words, when the pickup rate is higher than the return rate outside the designated service area, the additional benefit of using return reward to induce customers to return their bikes within the designated service area is very small, since the demand for bikes outside the designated service area combined with the pickup reward is sufficient to manage the number of bikes outside the designated service area.

Tables 3 and 4 present the corresponding results for the cases with $h = 1$ and $h = 10$, respectively. The main insights are similar to those illustrated in Table 2. However, with a higher value of h, we observe that even when both d_1 and d_0 are greater than d_2, the optimal dual reward scheme can provide substantial cost reduction over the optimal pickup reward scheme. This implies that return reward could still provide significant cost reduction in addition to pickup reward. In other words, when the operating cost of keeping bikes outside the designated service zone is high, the use of both types of rewards are useful in managing the amount of dockless bikes outside the designated service area. The pickup reward would serve as the primary incentive driver for increasing the number of customers to pick up bikes outside the designated service area, whereas the return reward can serve as a secondary incentive driver for reducing the number of bikes being returned outside the designated service area.

We note that the results of Tables 3 and 4 show that the use of pickup and/or return reward is beneficial even when pickup rate is much higher than return rate of bikes outside the designated service area; e.g., by comparing the results for the cases with $d_1 = 150 > d_2 = 100$ between the dual reward scheme and no reward scheme.

Table 3 Minimum cost under different incentive schemes with $s = 1$, $d_2 = 100$, and $h = 1$.

d_0	d_1	Dual Reward	Pickup Reward ($r_i = 0$)	Return Reward ($p_i = 0$)	No Reward ($r_i = p_i = 0$)
100	50	14.26	27.68	22.65	∞
	75	8.08	13.81	11.02	∞
	100	4.51	6.37	5.35	∞
	125	2.66	3.16	2.87	4.00
	150	1.74	1.89	1.80	2.00
500	50	7.38	8.91	22.65	∞
	75	4.90	5.69	11.02	∞
	100	3.24	3.62	5.35	∞
	125	2.20	2.37	2.87	4.00
	150	1.57	1.67	1.80	2.00
1000	50	5.22	5.76	22.65	∞
	75	3.72	4.03	11.02	∞
	100	2.65	2.83	5.35	∞
	125	1.93	2.05	2.87	4.00
	150	1.44	1.49	1.80	2.00
2000	50	3.64	3.84	22.65	∞
	75	2.77	2.90	11.02	∞
	100	2.12	2.19	5.35	∞
	125	1.64	1.68	2.87	4.00
	150	1.29	1.32	1.80	2.00

Table 4 Minimum cost under different incentive schemes with $s = 1$, $d_2 = 100$, and $h = 10$.

d_0	d_1	Dual Reward	Pickup Reward ($r_i = 0$)	Return Reward ($p_i = 0$)	No Reward ($r_i = p_i = 0$)
100	50	30.71	58.83	41.94	∞
	75	23.62	40.71	29.45	∞
	100	18.33	28.51	21.35	∞
	125	14.18	20.34	16.01	40.0
	150	11.41	14.93	12.43	20.0
500	50	18.70	23.34	41.94	∞
	75	15.67	19.08	29.45	∞
	100	13.17	15.63	21.35	∞
	125	11.12	12.89	16.01	40.0
	150	9.47	10.74	12.43	20.0
1000	50	14.14	16.21	41.94	∞
	75	12.28	13.89	29.45	∞
	100	10.67	11.92	21.35	∞
	125	9.31	10.27	16.01	40.0
	150	8.16	8.90	12.43	20.0
2000	50	10.45	11.38	41.94	∞
	75	9.36	10.12	29.45	∞
	100	8.39	9.01	21.35	∞
	125	7.54	8.05	16.01	40.0
	150	6.79	7.20	12.43	20.0

This suggests that the use of dynamic incentive schemes is always useful when there is a high cost of maintaining bikes outside the designated service, regardless of whether there is a proper balance between the return rate and demand rate of bikes outside the designated service area.

Table 5 Minimum cost under different incentive schemes with $s = 10$, $d_2 = 100$, and $h = 1$.

d_0	d_1	Dual Reward	Pickup Reward ($r_i = 0$)	Return Reward ($p_i = 0$)	No Reward ($r_i = p_i = 0$)
100	50	92.72	195.50	166.79	∞
	75	35.33	67.49	54.19	∞
	100	10.32	14.13	12.27	∞
	125	3.63	3.87	3.73	4.00
	150	1.96	1.99	1.97	2.00
500	50	39.37	48.35	166.79	∞
	75	18.15	21.48	54.19	∞
	100	7.41	8.16	12.27	∞
	125	3.34	3.44	3.73	4.00
	150	1.92	1.94	1.97	2.00
1000	50	25.05	27.68	166.79	∞
	75	12.78	13.81	54.19	∞
	100	6.05	6.37	12.27	∞
	125	3.10	3.16	3.73	4.00
	150	1.87	1.89	1.97	2.00
2000	50	15.81	16.56	166.79	∞
	75	8.95	9.27	54.19	∞
	100	4.90	5.01	12.27	∞
	125	2.81	2.84	3.73	4.00
	150	1.80	1.81	1.97	2.00

Tables 5 and 6 present the corresponding results of Tables 3 and 4 for the case with $s = 10$. The higher value of s corresponds to the situation where the return area outside the designated service area is geographically large such that the average travel distance required to pick up bikes outside the designated service zone area is higher. By comparing the corresponding results in Tables Tables 5 and 6 with those of Tables 3 and 4, we observe similar insights as described earlier, except that the minimum total expected cost is much higher under any of the optimal incentive schemes. This is due to the fact that, with a higher value of s, it would require a higher pickup/return reward to incentivize users to pick up or avoid returning bikes outside the designated service area due to a higher average travel distance. Overall, our numerical results suggest that the main insights are robust with respect to the value of s.

6 Concluding Remarks

We develop an analytical framework to evaluate the effectiveness of dynamic incentive schemes to balance supply and demand in a dockless bike-sharing system. Specifically, we explore the use of incentives for either picking up or returning bikes outside a certain designated service area. By modeling the number of bikes as a birth-and-death process, we can derive the system performance of the optimal dynamic incentive schemes in

Table 6 Minimum cost under different incentive schemes with $s = 10$, $d_2 = 100$, and $h = 10$.

d_0	d_1	Dual Reward	Pickup Reward ($r_i = 0$)	Return Reward ($p_i = 0$)	No Reward ($r_i = p_i = 0$)
100	50	143.2	280.8	226.4	∞
	75	80.8	139.5	110.2	∞
	100	45.1	63.8	53.5	∞
	125	26.6	31.6	28.7	40.0
	150	17.4	18.9	18.0	20.0
500	50	73.8	88.7	226.4	∞
	75	49.0	56.9	110.2	∞
	100	32.4	36.2	53.5	∞
	125	22.0	23.7	28.7	40.0
	150	15.7	16.5	18.0	20.0
1000	50	52.2	57.7	226.4	∞
	75	37.2	40.4	110.2	∞
	100	26.5	28.3	53.5	∞
	125	19.3	20.2	28.7	40.0
	150	14.4	15.0	18.0	20.0
2000	50	36.4	38.5	226.4	∞
	75	27.7	29.0	110.2	∞
	100	21.2	22.0	53.5	∞
	125	16.4	16.9	28.7	40.0
	150	12.9	13.3	18.0	20.0

which the incentive amount could vary depending on the number of bikes outside the designated service area.

Our results show that the optimal scheme should set a higher incentive amount as the number of bikes outside the designated service area increases in order to incentivize users to balance supply and demand in the system. Furthermore, it is sufficient to offer only a few incentive levels to achieve near-optimal performance. This result suggests that a dynamic, yet practical, incentive scheme can be useful in managing bikes in a dockless bike-sharing system. We further use a set of comprehensive numerical experiments to illustrate the specific operating conditions under which the use of such dynamic incentive schemes would be most beneficial in reducing the operating costs of the system.

References

1. Banerjee, S., Freund, D., Lykouris, T. (2017). Pricing and optimization in shared vehicle systems: an approximation framework. Working paper: *arXiv:1608.06819v3*.
2. Caggiani, L., Camporeale, R., and Ottomanelli, M. (2017). Planning and design of equitable free-floating bike-sharing systems implementing a road pricing strategy. *Journal of Advanced Transportation* Article ID 3182387, https://doi.org/10.1155/2017/3182387
3. Caggiani, L., Camporeale, R., Ottomanelli, M., Szeto, W.Y. (2018). A modeling framework for the dynamic management of free-floating bike-sharing systems *Transportation Research Part C: Emerging Technologies* **87**:159-182.
4. Chang, S., Song, R., He, S., Qiu, G. (2018). Innovative bike-sharing in China: Solving faulty bike-sharing recycling problem. *Journal of Advanced Transportation* Article ID 4941029, https://doi.org/10.1155/2018/4941029.
5. Cheng, X., Gao, Y. (2018). The optimal monthly strategy pricing of free-floating bike sharing platform. *Modern Economy* **9**(2):318338.

6. Chou, M.C., Liu, Q., Teo, C.P., Yeo, D. (2019). Models for effective deployment and redistribution of shared bikes with location choices. In *Sharing Economy* (edited by M. Hu), Springer Series in Supply Chain Management:409-434.
7. Freund, D., Henderson, S.G., Shmoys, D.B. (2019) Bike sharing. In *Sharing Economy* (edited by M. Hu), Springer Series in Supply Chain Management:435-459.
8. Fricker, C., Gast, N. (2016). Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. *EURO Journal on Transportation and Logistics* **5**(3):261-291.
9. Haider, Z., Nikolaev, A., Kang, J.E., Kwon, C. (2018) Inventory rebalancing through pricing in public bike sharing systems. *European Journal of Operational Research* **270**(1):103-117.
10. Jorge, D., Molnar, G., de Almeida Correia, G.H. (2015). Trip pricing of one-way station-based carsharing networks with zone and time of day price variations. *Transportation Research Part B: Methodological* **81**:461-482.
11. Pal, A., Zhang, Y. (2017). Free-floating bike sharing: Solving real-life large-scale static rebalancing problems. *Transportation Research Part C: Emerging Technologies* **80**: 92116.
12. Pfrommer, J., Warrington, J., Schildbach, G. (2014). Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems* **15**(4):1567-1578.
13. Pidd, H. (2018). Mobike no more: dockless bikes could soon be gone from UK streets. *The Guardian* (August 25).
14. Singla, A., Santoni, M., Bartok, G., Mukerji, P., Meenen, M., Krause, A. (2015). Incentivizing users for balancing bike sharing systems. *Proceedings of the 29th AAAI Conference on Artificial Intelligence*:723-729.
15. Waserhole, A., Jost, V. (2016). Pricing in vehicle sharing systems: optimization in queuing networks with product forms. *EURO Journal on Transportation and Logistics* **5**(3):293-320.
16. Waserhole, A., Jost, V., Brauner, N. (2013). Pricing techniques for self regulation in vehicle sharing systems. *Electronic Notes in Discrete Mathematics* **41**:149-156.
17. Xu, C., Ji, J., Liu, P. (2018). The station-free sharing bike demand forecasting with a deep learning approach and large-scale datasets. *Transportation Research Part C: Emerging Technologies* **95**:47-60.
18. Zhou, Z., Li, H., Zhang, A. (2019). Dockless sharing bikes, subway, and house prices: Evidence from micro-level data in Shanghai. Available at SSRN: https://ssrn.com/abstract=3409003 or http://dx.doi.org/10.2139/ssrn.3409003

MISTA 2019

Research on electroplating parallel-tank scheduling and optimization of product process selection in PCB manufacturing environment

Hao Wang • Lei Yue

1 Introduction

Printed circuit board (PCB) is an important electronic component for most of electronic devices and have a significant role in electronic manufacturing industries [1]. In PCB manufacturing system, the electroplating is one of the most critical processes, which contains a computer-controlled hoist to carry jobs for processing. The current research investigates a scheduling problem in electroplating process with identical parallel tanks and a constrained hoist, which comes from a real-world PCB enterprise in China. As shown in Fig. 1, products need to be moved in and out of the electroplating system by a constrained hoist. When the hoist is occupied by other tasks, products cannot enter or leave. In addition, some tanks will remain idle for if the processing sequence is not optimal, because the processing time of the products is different. Therefore, an optimized processing sequencing is very important to improve the output efficiency.

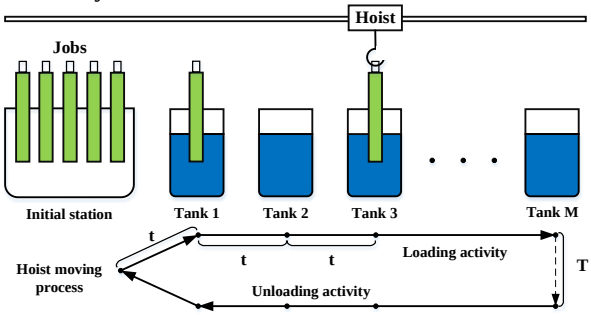


Fig.1. Configuration of the electroplating system and the hoist moving process

Hao Wang
The State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: skywang@hust.edu.cn

Lei Yue
The State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: leileyok@hust.edu.cn

2 Problem description

The electroplating system consists of *M* identical parallel tanks arranged in a row and a constrained hoist as shown in Fig. 1. There are *N* jobs waiting in the initial station. Firstly, the hoist loads one job to one of the idle tanks and moves to Tank *M* and stays there for a fixed period *T*. In the real-world PCB enterprise, the hoist is occupied by other tasks in this period. Later, the hoist moves back and unloads one of the finished jobs. After arriving the initial station, the hoist will start its next cycle. Finally, when the hoist takes out the last job, the processing is over.

The objective of the considered problem is to find an optimal processing sequence to minimize the makespan. The identical parallel machine scheduling problem is NP-hard [2]. The current problem contains hoist constraint, which is also NP-hard. Therefore, meta-heuristic algorithms are suitable for solving this problem.

3 Tabu search and comparison of different metaheuristics

3.1 The procedure of tabu search

The encoded mode is based on integer coding with *N* positive integers. A dynamic simulation process is formulated to describe the electroplating process logic and calculate the makespan. The initial solution is generated randomly and insert move is used to construct neighborhood solutions. Under the constraint of tabu list and aspiration criterion, the solution is constantly updated until the algorithm stops.

3.2 Experiment results

The performance of tabu search (TS) algorithm is compared with three common metaheuristics: genetic algorithm (GA), ant colony optimization (ACO) and particle swarm optimization (PSO). All algorithms are coded in Python and run on a computer with Intel Core i7, 3.1 GHz CPU, 16G RAM. The processing times of jobs are generated from discrete uniform (DU) distribution as DU[20,60], DU[60,100], DU[80,120] and DU[120,160]. The related electroplating system parameters are set using the actual data of the enterprise (i.e., T=4.5, t=0.05, M=15). As shown in Table. 1, the performance of TS is significant superior to ACO, PSO and GA, especially in cases with large processing times. Therefore, TS algorithm is effective in solving the proposed electroplating parallel-tank scheduling problem.

Table 1. Optimization results of TS, ACO, PSO and GA

Processing time	TS		ACO		PSO		GA	
	Min	Mean	Min	Mean	Min	Mean	Min	Mean
DU[20,60]	**222.0**	**222.0**	222.0	222.0	222.0	222.0	222.0	222.0
DU[60,100]	**264.0**	**264.0**	264.0	264.0	264.0	264.0	264.0	264.0
DU[80,120]	**288.5**	**289.4**	289.7	290.9	290.2	291.0	290.7	292.5
DU[120,160]	**370.8**	**372.2**	373.5	376.8	374.4	377.3	374.7	378.0

4 Optimization of product process selection

It can be observed from Table. 1, when processing time is generated from DU[20,60] and DU[60,100], the optimized values of makespan obtained by different algorithms are the same in each instance. This is because the processing time is too small, causing some tanks to be idle, while the hoist is running all time (the shortest hoist cycle time is T+2Mt = 6).

In the real-world PCB manufacturer, there are two processing modes (mode A and mode B) available for the products. The processing time of mode A is 40, while that of mode B can take values of 100, 120, 140, 150, etc. Too many jobs of mode A or mode B will cause tanks

or hoist to be idle for a long time. Therefore, it is necessary to find a proper ratio of mode A and B to optimize both the utilization of tanks and hoist. An experiment has been conducted to analyze the relationship between the ratio of A and B and the utilization of tanks and hoist.

First, a certain number of jobs of mode B (the number with processing time of 100, 120, 140 and 150 is 10 respectively) are given. Then, jobs of mode A are added one by one. Each time a job is added, the utilization of tanks and hoist will be obtained by running TS algorithm. Finally, the trend chart of utilization of tanks and hoist is obtained as shown in Fig. 2.

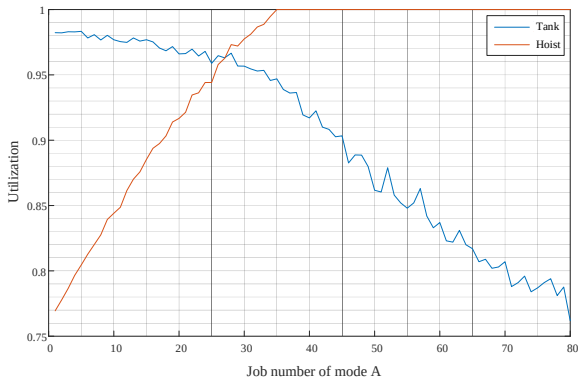


Fig. 2. Utilization of tanks and hoist under different ratios of A and B

In general, the utilization of hoist is increasing while the utilization of tanks is decreasing, as the job number of mode A increases. According to Fig. 2, after the job number of mode A is more than 35 (the average processing time of all jobs is 86.7), the hoist utilization is up to 100%, and the system overall efficiency continues to decline. Therefore, we mainly analyze the curves before the hoist utilization reaches 100%.

The same experiments are conducted on the different job numbers of mode B, and similar trend charts are obtained. Through curve fitting and setting weights for the utilization of tanks and hoist, it can be indicated that the system performance can be optimal when the average processing time of all jobs is close to 90. Therefore, the scheduler can select a better processing mode for different products with reference to this conclusion.

5 Conclusion

In conclusion, current research investigates an electroplating parallel-tank scheduling problem. Related experiment results indicate that TS algorithm can solve this problem effectively and obtain better solutions compared with ACO, PSO and GA. Further experiments verified that a proper ratio of product processing modes can improve the overall efficiency of electroplating system.

Acknowledgements This work was supported in part by the China Postdoctoral Science Foundation under Grant [2019M652665], and in part by the National Natural Science Foundation of China under Grant [51705379], Grant [51905196], Grant [71620107002], Grant [71131004], and Grant [71471071].

References

1. J. Mumtaz, Z. Guan, L. Yue, et al., Multi-level planning and scheduling for parallel PCB assembly lines using hybrid spider monkey optimization approach, IEEE Access, 7, 18685–18700 (2019)
2. D. N. Tahar, F. Yalaoui, C. Chu, et al., A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times, International Journal of Production and Economics, 99 (1), 63–73 (2006)

MISTA 2019

A pickup and delivery problem for the take-out delivery service

Xinrui Sun • Kunpeng Li

1 Introduction

With the development of mobile Internet, many customers tend to place an order online, which greatly promotes the development of takeout market based on O2O business model. At present, the size of the take-out market has reached more than 300 billion yuan. Maintaining a high level of delivery service quality is a key factor for takeout platforms to gain competitive advantage in the fierce competition.

In such takeout platforms, customers place orders with delivery service, choosing items from restaurants or shops. The delivery process of a take-out order is shown as follows. ① when a customer places an order with delivery service online, the take-out delivery system estimates the ready time of the corresponding merchant and the order's due date to the corresponding customer.②the system assign the order to an appropriate delivery clerk.③The delivery clerk pickups the order from the restaurants after ready time or shops and deliver it to the customer before the due date.

As multiple orders are placed, the delivery clerks are dispatched by the take-out delivery system. However, new orders arrive during the execution of the distribution plan. At some time, these new orders and committed orders having not been served should be reassigned to delivery clerks.

In this paper, we investigate a routing problem of the delivery clerks to serve new orders and committed orders when reoptimization occurring at some time. We refer to the problem as a pickup and delivery problem for the take-out delivery service, whose objective is to minimize the order's average distance. The aim of this paper is to present a linear mathematical formulation and a branch-and-cut algorithm to strengthen the model.

2 Problem description and mathematical formulation

We start by defining some notations presented in the mathematical formulation. There are n orders. A fleet of electric bicycles $H = \{1, \dots, K\}$ to implement the delivery task. $R = C_k \cup F$

Xinrui Sun
Huazhong University of Science and Technology, School of Management
E-mail: 1178745380@qq.com

Kunpeng Li
Huazhong University of Science and Technology, School of Management
E-mail: likp@mail.hust.edu.cn

denotes an order set, where C_k represents the committed orders that have been assigned to electric bicycle k, F represents the flexible orders (new orders), respectively.

We are given a complete directed graph $G = (N, A)$. The vertex set $N = \{0\} \cup C_k^+ \cup C_k^- \cup F^+ \cup F^- \cup M$, where C_k^+ represents the set of all merchants referred to C_k, C_k^- represents the set of all customers referred to order set C_k, F^+ denotes the set of all merchants referred to order set F, F^- denotes the set of all customers referred to order set F. $M = \{\mu_k | k \in H\}$, where μ_k denotes the current location of the electric bicycle k. Additionally, node 0 denotes the distribution station.

$A = \{(i,j) | i,j \in N\}$ is an arc set. Each arc $(i,j) \in A$ is associated with non-negative coefficients d_{ij} and t_{ij}, where d_{ij} represents the distance between node i and node j, t_{ij} denotes the travel time between node i and node j.

For the convenience of modelling, each order $r \in R$ refers to a merchant node $i_r \in C_k^+ \cup F^+$,a customer node $n + i_r \in C_k^- \cup F^-$and demand q_r. rt_{i_r}denotes the ready time required by the merchant i_r.dt_{n+i_r} denotes the due date that the order is delivered to customer $n + i_r$. Additionally, $q_{i_r} = q_r$ denotes the demand of node i_r, $q_{n+i_r} = -q_r$ represents the demand of node $n + i_r$.

The problem is to construct $|H|$ routes to minimize the average distance of orders such that:

(1) Each electric bicycle must serve the committed orders previously assigned to it;
(2) Each order is served by one electric bicycle;
(3) For each order, the delivery clerk should pickup it from corresponding merchant after the ready time, and deliver it to corresponding customer before related due date.
(4) The quantity of each electric bicycle should not exceed its capacity.

We define decision variables:

x_{ij}^k, equals 1, if electric bicycle k travels from node i to node j; otherwise,0.
Q_i^k , the loaded quantity when electric bicycle k visits node i.
t_i^k, the time that electric bicycle k serve node i.

The proposed formulation is as follows.

$$Minimize \frac{\sum_{k \in H} \sum_{i \in N} \sum_{j \in N} d_{ij} * x_{ij}^k}{n} (1)$$

$$\sum_{\mu_k, j} x_{\mu_k, j}^k = 1, \forall k \in H, \forall j \in N \backslash (\{0\} \cup M) (2)$$

$$\sum_{i \in \{\mu_k\} \cup C_k \cup F} x_{ij}^k = 1, \forall k \in H, \forall j \in C_k \ (3)$$

$$\sum_{k \in H} \sum_{i \in N \backslash \{0\}} x_{ij}^k = 1, \forall k \in H, \forall j \in F (4)$$

$$\sum_{i \in N} x_{ij}^k = \sum_{i \in N \backslash \{\mu_k\}} x_{ji}^k, \forall k \in H, \forall j \in N (5)$$

$$0 \leq Q_i^k \leq Q, \forall i \in N, \forall k \in H (6)$$

$$Q_j^k \geq Q_i^k + q_i - M * \left(1 - x_{ij}^k\right), \forall i, j \in N, \forall k \in H (7)$$

$$t_i^k \geq rt_i * (\sum_{j \in N} x_{ji}^k), \forall i \in C_k^+ \cup F^+, \forall k \in H (8)$$

$$t_i^k \leq dt_i * (\sum_{j \in N} x_{ji}^k), \forall i \in C_k^- \cup F^-, \forall k \in H (9)$$

$$\sum_{j \in N} x_{ji}^k = \sum_{j \in N} x_{j,n+i}^k, \forall i \in C_k^+ \cup F^+, \forall k \in H \ (10)$$

$$t_{n+i}^k \geq t_i^k, \forall i \in C_k^+ \cup F^+, \forall k \in H (11)$$

$$t_j^k \geq t_i^k + t_{ij} - M * \left(1 - x_{ij}^k\right), \forall i, j \in N, \forall k \in H (12)$$

Constraint (1) is to minimize the average distance of orders. Constraints (2) ensure that each delivery clerk departs from his current location. Constraints (3) are that each electric bicycle k must serve the committed orders previously assigned to it. Constraints (4) guarantee that each node is visited only once by one electric bicycle. Constraints (5) make sure the flow balance. Constraints (6)-(7) are that each electric bicycle's capacity is respected at all nodes. Constraints (8) specify that each order is available to be pickup from corresponding merchant after its ready time; Constraints (9) specify that each order is delivered to corresponding customer before due date. Constraints (10) force each order is served by one electric bicycle. Constraints (11) indicate the time limit for each order. Constraints (12) are that the serve time of each node is respected.

3 a branch-and-cut algorithm

This section briefly describes the proposed branch-and-cut algorithm over the mathematical formulation. The solution contains four sections:①Preprocessing: To speed up the speed of solving the problem, we eliminate some redundant arcs based the problem characteristics;② Separation of strategy: We solve the relaxation of the model, and receive a current solution. Then we design related separation procedure to identify violated inequalities based current solution for each proposed valid inequality;③Node Selection : We use best-bound-first rule to select node.④Branch strategy: According to current fraction solution, we select a variable x_{ij}^k that satisfy $|\left(x_{ij}^k\right)^* - 1|$ is smallest as a branch variable.

References

1. P. Chen, H. Li, Optimization Model and Algorithm based on the Time Satisfaction for O2O Food delivery, Chinese Journal of Management Science, (2016).
2. Z. Steever, M.Karwan,C. Murray, Dynamic courier routing for a food delivery service, Computers and Operations Research, 107 (2019) 173-188.
3. M. Cosmi, G. Oriolo, V. Piccialli, P. Ventura, Single courier single restaurant meal delivery (without routing), Operations Research Letters, 47 (2019) 537-541.
4. G. Ninikas, I. Minis, Reoptimization strategies for a dynamic vehicle routing problem with mixed backhauls, (2014).
5. P. Chen, H. Li, Optimization Model and Algorithm based on the Time Satisfaction for O2O Food delivery, Chinese Journal of Management Science, (2016).

MISTA 2019

An Optimized Analysis of Crew Scheduling Plan for Railway Operation

Lee, JaeHee • Park, Sangmi • An, Hyeonju• Kang, Leenseok

1 Introduction

The crew diagram in the railway operation institutes means an individual service schedule that expresses the train number which a crew member must operate for a day, the name of the station passing through, the operation time, the rest time, and the transfer time. All crew members can use the diagram to identify the trains that need to be operated on that day and the names of the stations that need to be moved. Such a crew diagram should be planned in consideration of the working hours of the crew, the boarding time, and the rest time, which are defined in the working conditions of the operating institution. The crew roster means a crew operating table that shows the train number and boarding time that several crew members belonging to the railway operating organization should ride on a daily basis. When several crew members are working in an operating institution, the working hours between crew members should have a small standard deviation and a crew roster should be prepared so that the number of weekly working days is evenly distributed.

Since most of crew schedules are written by expert's empirical knowledge, it is time-consuming and difficult to determine whether they are optimal solutions. In this study, a methodology is suggested to enable efficient management of crew schedule to be worked by applying a neighborhood search method. The crew diagram using the proposed methodology is proving to be efficient by comparing the results with the actual train timetables.

Tormos, P et al. (2008) applied GA (Genetic Algorithm) as a solution to train operation planning problems, and Khmeleva, E et al. (2014) also proposed a genetic algorithm as a solution for scheduling crew roster of freight train crews. Existing researches to solve crew schedule problems include neighborhood search method, integer programming, and heuristic repair methods. Integer programming is effective when both constraint and objective function are expressed linearly. The result of genetic algorithms can vary greatly depending on how they are expressed in gene form. In this study, the users set variable values and add constraints in the initial stage of crew diagram generation to narrow the search range. By doing this, the user finds the optimal combination in a relatively short time.

Lee, Jaehee
Department of Civil Engineering, Gyeongsang National University
E-mail: akfn1614@gmail.com

Park, Sangmi
Department of Civil Engineering, Gyeongsang National University
E-mail: gogf8585@naver.com

An, Hyeonju
Department of Civil Engineering, Gyeongsang National University
E-mail: guswn757@naver.com

Kang, Leenseok
Professor, Department of Civil Engineering, Gyeongsang National University
E-mail: lskang@gnu.ac.kr

2 Analysis Process for Optimized Solution for Crew Scheduling

The crew diagram makes it easy to see the operating day, time of commute and work, total crew time, one-way boarding time, succession waiting time, and total working time. Those times can be easily identified from the crew diagram. The number of crew diagrams run by S train operating institution which are used as an example in the study was surveyed as 347 times in a day time, 134 times at a night time, during a week. Among them, the crew diagrams of one-way boarding were 52times. One-way boarding is a factor that hinders manpower operation efficiency from the point of the train operating institution, so it is necessary to minimize it. The crew diagram is generated by linking trains together that meet the constraints. Moreover, in this process, all possible running trains must be paired in order to minimize take riding business. The model for creating a crew diagram using the neighborhood search method is shown in Figure 1.

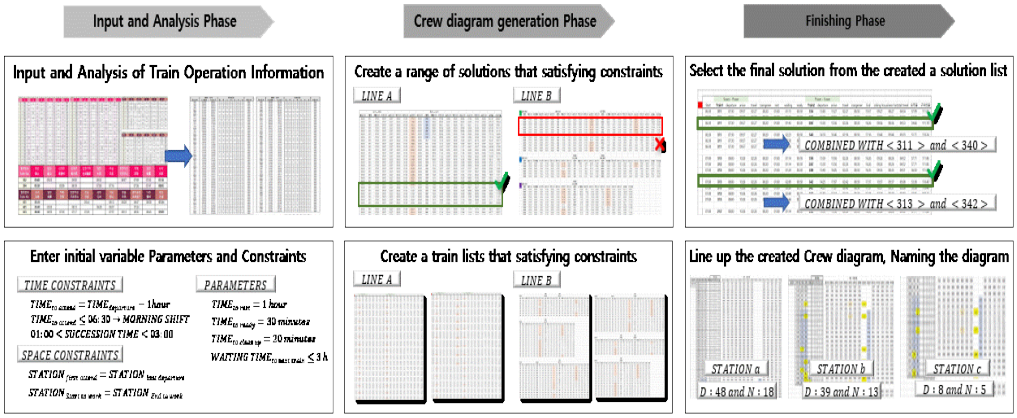


Figure 1. The Procedure of generating a crew diagram using neighborhood search method

In the crew diagram generation phase, a list of possible combinations of crew diagrams are created by combining data of train operation reclassified according to operation day, route, and duty type. In the finishing phase, select the final crew diagram solution from the list of diagram generation. After sorting this, the user completes the creation of the crew diagram by naming the crew center and route.

The daytime diagram is generated by combining the trains that satisfy the constraints. The constraints to be considered are the temporal constraints and the spatial constraints. After end of the first train trip, operating trains that meet the time constraints (20 minutes of cleanup time, 1 hour of rest time and 30 minutes of succession waiting) may be combined into the next train trip. In addition, the diagram is must be satisfied the spatial constraints, so that the crew member can finish the trip at the same station where they started. According to these temporal and spatial constraints, the search range of trains that can be combined into crew diagrams are reduced.

The generated crew diagram is moved to the crew rostering phase and assigned to the crew members of each station center appropriately. After this process, the monthly schedule of the crew is completed. In the crew rostering phase, the crew diagram should be arranged appropriately to satisfy the Labor Standard Act and in-house labor regulations. Figure 2 shows the process of assigning the day and night crew diagrams to the crew.

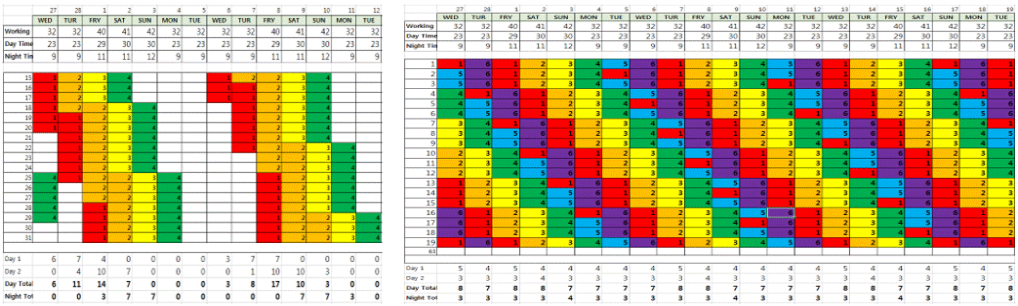


Figure 2. The procedure of assigning diagram to the crews

3 Verification and Conclusions

In order to verify the analysis results of the neighborhood search method attempted in the study, the number of crew diagrams in the train schedule, the average value of major times and the crew ratio were compared. The results show that the average business hours of two crew center diagrams increased during the day and night, but the standard deviation decreased by more than 30% on average. Table 1 shows the result of comparison among the crew diagrams generated by the methodology suggested in this study and those of the existing S train operating institution. The number of crew diagrams for daytime and nighttime in station A was decreased. Station B was reduced by 1 daytime diagram and increased by 2 nighttime diagrams. Station C was increased by 6 daytime diagrams and decreased by 1 nighttime diagram. Take riding was reduced from 52 to 39 in total. The standard deviation of the crew diagram is reduced except for the crew diagram at station B.

Table 1. Comparison of crew diagram

		Station							
		A		B		C		Total	Take riding
The number of week crew diagrams	Existing crew diagram	D	N	D	N	D	N	481	52
		181	70	116	41	50	23		
	Generated crew diagram	D	N	D	N	D	N	474	39
		174	64	115	43	56	22		
The standard deviation of crew diagram	Existing crew diagram	45 min		35 min		75 min		-	-
	Generated crew diagram	28 min		35 min		57 min		-	-
	Result	Decrease		No change		Decrease		-	-

In this study, crew diagram was generated by applying a neighborhood method for efficient management of crew schedule. The efficiency of the methodology was verified by comparing with existing crew diagram. The reduction of one-way boarding diagrams and standard deviations can improve the crew manpower efficiency in the crew rostering process. According to the method proposed in this study, variable values determined by the user can be input differently according to different crew schedule planning methods and constraints, and as a result, the combination method and solution search range are changed. As such, the search for a neighborhood search method that satisfy the constraint may be one of many methods for solving the problem of preparing a crew.

Acknowledgements

The authors would like to thank the Ministry of Land, Infrastructure and Transport of Korea for financially supporting a part of this research under 2019 R&D program project (19RTRP-Cl48789-02).

References

1. Khmeleva, E, Hopgood, A.A, Tipi, L, and Shahidan, M, Rail-Freight Crew Scheduling with a Genetic Algorithm, Research and Development in Intelligent system, 31, 211-223 (2014)
2. Tormos, P, Lova, A, Barber, F, Ingolotti, L, Abril, M and Salido, M.A, A Genetic Algorithm for Railway Scheduling Problems, Methaheuristics for Scheduling in Industrial and Manufacturing Applications : Part of the Studies in Computational Intelligence, 128, 255-276 (2008)

MISTA 2019

Multi-objective dynamic programming based on fuzzy optimal selection model and its application to disassembly sequence planning

Yusha Zhou •Xiuping Guo

1 Introduction and problem statement

Design for end-of-life and design for disassembly are enabling design strategies for the implementation of business models based on the circular economy paradigm [1]. Disassembly sequence planning (DSP) is a major problem in the disassembly planning problem and is known to be NP-hard. A considerable amount of literature of DSP has been published, these studies mainly focus on nature-inspired heuristic algorithms, linear programming methods, rule-based methods, stochastic simulation techniques [2-5]. However, the use of fuzzy optimization model has not been investigated. In this paper, a hybrid methodology that combines fuzzy optimization model with multi-objective dynamic programming, which relies on conventional dynamic programming, is proposed to solve DSP. A characteristic feature of the approach proposed is that multiple objectives are synthetically considered by the fuzzy optimization technique instead of the frequently employed linear weighted sum method. We consider two objectives in the same time, i.e., minimizing the operating time and the operating costs. The two objectives concern the operating cost to perform the disassembly tasks, which are typically manual and labor intensive. The disassembly task costs and times are dependent upon the disassembly tools, operating orders and worker's skill. Therefore, the costs and times incurred could be quite different between disassembly sequences. It mainly considers a single type of product that undergoes a number of ordered tasks for complete disassembly. Each task leads to the removal of one part except the final task that results in the separation of two parts. There may exist several different disassembly sequences or routes. We use an example to illustrate the relationship between the disassembly sequences and the Transformed AND/OR Graphs (TAOG) [6]. As shown in Figure 1, the original product is represented by A_0, the artificial nodes (A_i) represent subassemblies and the normal nodes (B_j) represent disassembly tasks in the TAOG. Note that we have inserted a final artificial node A_{12} to represent that the disassembly is completed. The multiple and alternative precedence relations are represented by the OR-type which is indicated with a small curve in Figure 1. That means a subassembly could be disassembled via alternative tasks. The disassembly procedure starts with a task of

Yusha Zhou
Southwest Jiaotong University
E-mail: yszhou@my.swjtu.edu.cn

Xiuping Guo
Southwest Jiaotong University
E-mail: gxp@swjtu.edu.cn

either B_1, B_2 or B_3 and ends with a task of either B_{19}, B_{20}, B_{21} or B_{22}. For example, A_0-B_3-A_3-B_9-A_6-B_{15}-A_9-B_{20}-A_{12} is feasible route which start with B_3. And the product is disassembled completely. So our aim is to find optimal route which meets precedence relationship among the tasks.

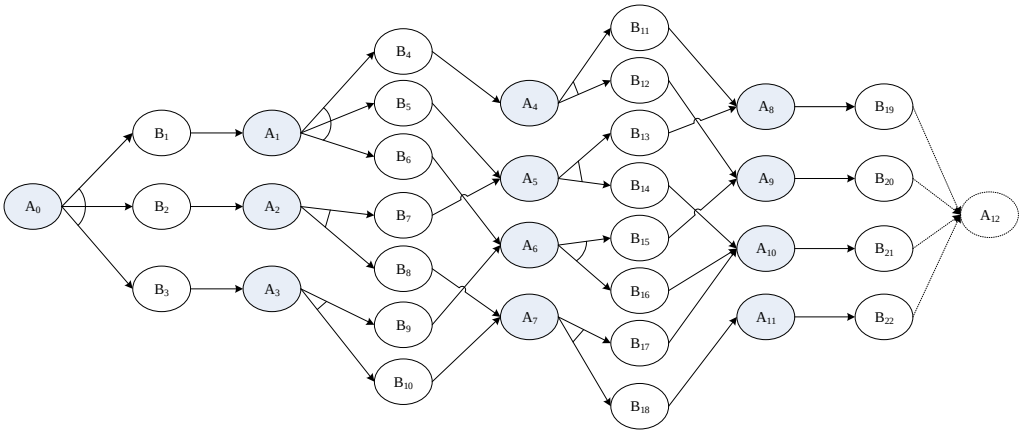


Figure 1: An example of the TAOG

2 Solution approach

The approach chosen to solve the problem considered is based on fuzzy optimum theory and the concept of dynamic programming and is a multi-stage fuzzy decision-making method [7-9]. Two methods to solve the problem are as follows:

- The maximum sum of relative membership degree

$$X_k = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}_k = \left(x_{ij}^k\right)$$

Where X_k is the target eigenvalue matrix at stage k, x_{ij}^k represents the target value of objective ith $(i = 1, \cdots, m)$ when decision x_j $(j = 1, \cdots, n)$ is made at stage k. The relative membership degrees of the elements of matrix X_k are computed by follows:

$$r_{ij}^k = 1 - \frac{x_{ij}^k}{\max\limits_{j,k=1,\cdots N} x_{ij}^k}$$

Then matrix X_k can be transformed into the relative membership degree matrix:

$$R_k = \begin{pmatrix} r_{11}^k & \cdots & r_{1n}^k \\ \vdots & \ddots & \vdots \\ r_{m1}^k & \cdots & r_{mn}^k \end{pmatrix} = \left(r_{ij}^k\right)$$

Assume that objectives have different weights, so the weight vector is as follows:

$$\mathbf{w} = (w_1, w_2, \ldots, w_m)$$

Which is subject to $w_i \geq 0$ and $\sum_{i=1}^m w_i = 1$

The fuzzy optimization model is as follows:

$$u_j^k = \frac{1}{1 + \frac{\sum_{i=1}^m \left[w_i^k (1 - r_{ij}^k)\right]^2}{\sum_{i=1}^m (w_i^k r_{ij}^k)^2}} = \phi_k(s_{k-1}, x_k)$$

Then by the optimal principle of conventional dynamic programming, we have the backward recursive relations

$$\varphi_k^\star(s_{k-1}) = \begin{cases} \max\limits_{x_k}\{\phi_k(s_{k-1}, x_k) + \varphi_{k+1}^\star(s_k)\} & k = N-1, \cdots, 1 \\ 0 & otherwise \end{cases}$$

Our objective is to find the optimal value of $\varphi_1^\star(s_0)$, which can be calculated by solving the above recursive equations. At k = 1, $\varphi_1^\star(s_0) = \varphi_1^\star(x_1^\star)$ is answer desired. The fuzzy optimal decision sequence $\{x_1^\star, x_2^\star, \dots, x_{N-1}^\star\}$ can be traced by $x_k^\star = x^\star(s_k^\star)$, k= 1,2,…,N-1 with the state transition equation

$$s_1^\star = s_0,\ s_{k+1}^\star = f(x_k^\star, s_k^\star),\ \text{k=1, …, N-1.}$$

- Stage fuzzy optimum selection method

If backward recurrence to stage k, the state is $s_{k-1,h}$, satisfying certain constraints, having admissible decision x_j with the number of n_t. After taking action $x_j \in U(s_{k-1,h})$, the state $s_{k-1,h}$ transits into state $s_{k,h}$. The merits of each decision are judged by the eigenvalues of M objectives. For stage k, state $s_{k-1,h}$, making decision x_j, its target eigenvalue vector is

$$\boldsymbol{x_j^k}(s_{t-1,k}, x_j) = (x_{1j}, x_{2j}, \dots, x_{mj})$$

Then the target eigenvalue matrix of admissible decision x_j with the number of n_t is expressed as $\boldsymbol{X^k}(s_{k-1,h})_{m\times n_t}$.

$$\boldsymbol{X^k}(s_{k-1,h})_{m\times n_t} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n_t} \\ x_{21} & x_{22} & \cdots & x_{2n_t} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn_t} \end{pmatrix}$$

Where h=1, 2, … , n_{t-1}; n_{t-1} is the number of state at the stage k.

Indicates that n_t decisions alternatively are made under the condition that the state $s_{k-1,h}$ is known: $x_1, x_2, \dots, x_{n_t}$, and correspondingly the n_t state of stage k, namely $s_{k1}, s_{k2}, \dots, s_{kn_t}$. Each state s_{kj} corresponds to an optimal multi-objective eigenvalue vector at the later stage k, which is denoted by $_{\boldsymbol{k+1}}^{\star}\boldsymbol{f_j}(s_{kj})$, namely

$$_{\boldsymbol{k+1}}^{\star}\boldsymbol{f_j}(s_{kj}) = \left(f_{1j}^\star(s_{kj}), f_{2j}^\star(s_{kj}), \dots, f_{mj}^\star(s_{kj})\right) \quad \text{j=1,2, … , } n_t.$$

$$_{\boldsymbol{k+1}}^{\star}\boldsymbol{F}(s_{kj})_{m\times n_t} = \begin{pmatrix} f_{11}^\star(s_{k1}) & f_{12}^\star(s_{k2}) & \cdots & f_{1n_t}^\star(s_{kn_t}) \\ f_{21}^\star(s_{k1}) & f_{22}^\star(s_{k2}) & \cdots & f_{2n_t}^\star(s_{kn_t}) \\ \vdots & \vdots & \vdots & \vdots \\ f_{m1}^\star(s_{k1}) & f_{m2}^\star(s_{k2}) & \cdots & f_{mn_t}^\star(s_{kn_t}) \end{pmatrix}$$

Matrix $_{\boldsymbol{k+1}}^{\star}\boldsymbol{F}(s_{kj})_{m\times n_t}$ is the local optimal multi-objective matrix in the backward stage. Then

$$\boldsymbol{F^k}(s_{k-1,h})_{m\times n_t} = \boldsymbol{X^k}(s_{k-1,h})_{m\times n_t} \otimes {}_{\boldsymbol{k+1}}^{\star}\boldsymbol{F}(s_{kj})_{m\times n_t} = \begin{pmatrix} f_{11}(s_{k-1,h}) & f_{12}(s_{k-1,h}) & \cdots & f_{1n_t}(s_{k-1,h}) \\ f_{21}(s_{k-1,h}) & f_{22}(s_{k-1,h}) & \cdots & f_{2n_t}(s_{k-1,h}) \\ \vdots & \vdots & \vdots & \vdots \\ f_{m1}(s_{k-1,h}) & f_{m2}(s_{k-1,h}) & \cdots & f_{mn_t}(s_{k-1,h}) \end{pmatrix}$$

The compound matrix $\boldsymbol{F^k}(s_{k-1,h})_{m\times n_t}$ represents n_t eigenvalues vectors of m objective functions of state $s_{k-1,h}$ at stage k.

Exclude dominant target vectors in $\boldsymbol{F^k}(s_{k-1,h})_{m\times n_t}$, save non-inferior target vectors, denoted by $\boldsymbol{F^k}(s_{k-1,h})_{m\times q}$, q ≤ n_t. Applying the Formula of Relative membership degree,

$$r_{ij}^k(s_{k-1,h}) = 1 - \frac{f_{ij}(s_{k-1,h})}{\max\limits_j f_{ij}(s_{t-1,h})} \quad \text{for i} = 1,2,\cdots,\text{m}$$

The matrix $\boldsymbol{F^k}(s_{k-1,h})_{m\times q}$ is transformed into the relative membership matrix of the target vectors:

$$\boldsymbol{R^k}(s_{k-1,h})_{m\times q} = \begin{pmatrix} r_{11}^k(s_{k-1,h}) & r_{12}^k(s_{k-1,h}) & \cdots & r_{1n_t}^k(s_{k-1,h}) \\ r_{21}^k(s_{k-1,h}) & r_{22}^k(s_{k-1,h}) & \cdots & r_{2n_t}^k(s_{k-1,h}) \\ \vdots & \vdots & \vdots & \vdots \\ r_{m1}^k(s_{k-1,h}) & r_{m2}^k(s_{k-1,h}) & \cdots & r_{mn_t}^k(s_{k-1,h}) \end{pmatrix} = r_{ij}^k(s_{k-1,h})$$

Application fuzzy optimization model

$$u_j^k(s_{k-1,h}) = \frac{1}{1 + \frac{\sum_{i=1}^m [w_i^k(1 - r_{ij}^k(s_{k-1,h}))]^2}{\sum_{i=1}^m (w_i^k \cdot r_{ij}^k(s_{k-1,h}))^2}}$$

According to the principle of maximum membership degree, that is, according to **max**$\{\ \boldsymbol{u_j^k(s_{k-1,h})}\}$**, j=1,2,…,** $\boldsymbol{n_t}$**,** a locally optimal multi-objective eigenvalue vector for the initial state $s_{k-1,h}$ of stage k can be obtained:

$${}_k^\star f_j(s_{k-1,h}) = \left({}_k^\star f_{1j}(s_{k-1,h}), {}_k^\star f_{2j}(s_{k-1,h}), \ldots, {}_k^\star f_{mj}(s_{k-1,h})\right)$$

Similar calculations are made for the state $s_{k-1,h}$ (h=1,2,.., n_{t-1}). The backward local optimal multi-objective matrix of k stage is obtained.

$${}_k^\star F(s_{k-1,p})_{m\times n_{t-1}} = \begin{pmatrix} {}_k^\star f_{11}(s_{k-1,1}) & {}_k^\star f_{12}(s_{k-1,2}) & \cdots & {}_k^\star f_{1n_{t-1}}(s_{k-1,n_{t-1}}) \\ {}_k^\star f_{21}(s_{k-1,1}) & {}_k^\star f_{22}(s_{k-1,2}) & \cdots & {}_k^\star f_{2n_{t-1}}(s_{k-1,n_{t-1}}) \\ \vdots & \vdots & \vdots & \vdots \\ {}_k^\star f_{m1}(s_{k-1,1}) & {}_k^\star f_{m2}(s_{k-1,2}) & \cdots & {}_k^\star f_{mn_{t-1}}(s_{k-1,n_{t-1}}) \end{pmatrix}$$

i=1,2,…,m; p=1,1,…, n_{t-1}.

$$\boldsymbol{{}_k^\star F}(s_{k-1,p})_{m\times n_{t-1}} = F \cdot Opt_{h=1,2,..,n_{t-1}} \left\{ \boldsymbol{X^k}(s_{k-1,h})_{m\times n_t} \otimes \boldsymbol{{}_{k+1}^\star F}(s_{kp})_{m\times n_t} \right\}$$

That is, $\boldsymbol{{}_k^\star F}(s_{k-1,p})_{m\times n_{t-1}}$ can be solved by the fuzzy optimization model from the backward recursive matrix.

3 Summary

The present study is intended to deal with DSP by the proposed approach. Selection is made based on fuzzy optimal selection model at each stage, fuzzy optimal selection operation will reduce the computing time for the application in products with more parts. As for future work, the effectiveness and superiority of the proposed methods will be verified by a series of test examples. Moreover, a variety of disassembly schemes will be presented in a practical disassembly case. The substantially optimal plan obtained is quite suitable, because it admits human interpretation and interference. In this regard, fuzzy optimal selection for multi-

objectives DP is not only a mathematical tool but also a decision support system for management.

Acknowledgements

This work is supported by the National Natural Science Foundation of China [grant number 71471151].

References

[1] Marco Marconi, Michele Germani, Marco Mandolini & Claudio Favi, Applying data mining technique to disassembly sequence planning: a method to assess effective disassembly time of industrial products, International Journal of Production Research, 57:2,599-623 (2019).
[2] Zhou Z., J. Liu, D.T. Pham, W. Xu, F.J. Ramirez, C. Ji, Q. Liu, Disassembly sequence planning: recent developments and future trends, Proc IMechE Part B: J Engineering Manufacture 1–22 (2018).
[3] Liu, J., Z. Zhou, D.T. Pham, W. Xu, C. Ji, Q. Liu, Collaborative optimization of robotic disassembly sequence planning and robotic disassembly line balancing problem using improved discrete Bees algorithm in remanufacturing. Robotics and Computer-Integrated Manufacturing, 61:101829 (2020).
[4] Tseng, H.E., C.C. Chang, S.C. Lee, Y.M. Huang, Hybrid bidirectional ant colony optimization (hybrid BACO): An algorithm for disassembly sequence planning. Engineering Applications of Artificial Intelligence, 83: 45-56 (2019).
[5] Ren, Y.P., C.Y. Zhang, F. Zhao, H.J. Xiao, G.D. Tian, An asynchronous parallel disassembly planning based on genetic algorithm. European Journal of Operational Research, 269(2): 647-660 (2018).
[6] Koc, A., I. Sabuncuoglu, and E. Erel, “Two Exact Formulations for Disassembly Line Balancing Problems with Task Precedence Diagram Construction Using an AND/OR Graph.” IIE Transactions 41(10):866-881 (2009).
[7] Mohammad L. Hussein, Mahmoud A. Abo-Sinna. A fuzzy dynamic approach to the multi-criterion resource allocation problem, Fuzzy Sets and Systems, 69:115 -124(1995).
[8] Chen Shouyu, Theory of fuzzy selection for multistage and multiobjective decision making system, The Journal of Fuzzy Mathematics, 2(1) (1994).
[9] Mahmoud A. Abo-Sinna. Multiple objective (fuzzy) dynamic programming problems: a survey and some applications, 157:861–888 (2004).
[10] Lushu Li, K.K. Lai, Fuzzy dynamic programming approach to hybrid multiobjective multistage decision-making problems, Fuzzy Sets and Systems 117 (2001) 13-25.

MISTA 2019

Online Supply Chain Scheduling Problems

Zhi-Long Chen, University of Maryland, College Park, Maryland, USA / Email: zchen@rhsmith.umd.edu
Lixin Tang, Northeastern University, Shenyang, China / Email: lixintang@ise.neu.edu.cn
Feng Li, Huazhong University of Science & Technology, Wuhan, China / Email: fengli055@gmail.com

Extended Abstract

1. Problems Studied

Supply chain scheduling problems have received a lot of attention in the last 15 years or so. However, majority of the existing research on such problems deals with offline problems where all the information about a job is known to the decision maker in advance. Relatively, much less work has been done on online supply chain scheduling problems where jobs arrive randomly over time and the decision maker does not know the information of a job until it arrives.

In this talk, we consider online supply chain scheduling involving processing of jobs and two-stage job delivery of completed jobs. The jobs that arrive over time are first processed in a plant with one or m production lines, and then delivered to one or k customer sites K through two stages of shipping. In the first shipping stage, jobs are delivered from the plant to a pool point (e.g., a port, a distribution or a consolidation center) by homogeneous transporters (e.g., trailers, vans, or trucks), which we call *vehicles* to distinguish them from transporters used in the second stage of delivery. We assume that vehicles are always available whenever needed. Each vehicle can deliver up to c_0 jobs in each shipment. Jobs from different customers can be delivered together in a shipment by a vehicle. The transportation time and transportation cost of each shipment by a vehicle from the plant to the pool point are t_0 and f_0, respectively. In the second shipping stage, jobs are delivered from the pool point to the customers by transporters (e.g., ships, trucks, or vans) which we call *ships* to distinguish them from the transporters used in the first stage of delivery. Ships are generally heterogeneous with different capacities and customer sites. However, the ships going to the same customer site are homogeneous. We assume that ships for each customer site are always available whenever needed. Each ship going to customer $i \in K$ can carry up to c_i jobs in each shipment. Transportation time t_i and cost f_i are associated with a shipment from the pool point to customer $i \in K$. For ease of presentation, the shipments by vehicles from the plant to the pool point and the shipments by ships from the pool point to the customers are denoted below as *v-shipments* and *s-shipments*, respectively. The transportation times from the plant to the pool point and from the pool point to the customer sites are denoted as *v-times* and *s-times*, respectively.

We must determine (1) how to schedule jobs for processing on the production lines of the plant; (2) how to batch jobs to form v-shipments and schedule the delivery of v-shipments from the plant to the pool point; (3) how to batch jobs to form s-shipments and schedule the delivery of s-shipments from the pool point to the customers, so as to optimize a performance measure that takes into account both delivery timeliness and total transportation costs. We consider two problems in which delivery timeliness is measured by total or maximum lead time of the jobs and denote them as P1 and P2, respectively.

2. Results Developed

For problem P1 with a single customer and a single production line, i.e., $k = 1$ and $m = 1$, we propose an online algorithm with a competitive ratio of 4, and for the general case with multiple customers and multiple production lines, i.e. $k \geq 1$ and $m \geq 1$, we propose an online algorithm with a competitive ratio of $c_{max} + 1$. Since our problem P1 is more general than all existing online integrated production and distribution scheduling problems with a similar objective function, our results apply to these problems and their generalizations with two stages of delivery and with parallel machines at the plant.

For problem P2, for the case with a single customer, i.e., $k = 1$, we propose an optimal online algorithm with a competitive ratio of 2, and for the general case with multiple customers, i.e. $k \geq 1$, we propose an online algorithm with a competitive ratio of $k + 2$.

3. Related Literature

To the best of our knowledge, all the existing online scheduling problems with job delivery involve a single delivery stage only. Thus, our research is the first attempt on online scheduling problems with two delivery stages. In addition, in all the existing online integrated production and delivery scheduling problems with the total delivery time of jobs as a part of the objective function, there is only a single machine at the plant; and in all the existing online integrated production and delivery scheduling problems involving batch delivery with the maximum delivery time of jobs as a part of the objective function, there is only a single customer. Thus, our research extends the current literature on online integrated production and delivery scheduling problems in several directions.

For the online single-machine scheduling problem with the objective of minimizing total completion time, which is a special case of our problem P1 with zero delivery time and cost, Hoogeveen and Vestjens (1996) and Lu et al. (2003) present optimal 2-competitive algorithms. For the same problem but with m identical parallel machines, Lu et al. (2003), Liu and Lu (2009) and Sitters (2010) provide on-line algorithms with competitive ratio of 2β, 2, and $1.79 + o(m)$, respectively, where β is the performance ratio of the shortest remaining processing time first rule for the preemptive relaxation of the problem. More recently, Feng et al. (2016) study two cases of an online integrated production and one-stage delivery scheduling problem with a single machine and the same objective function as in our problem P1. For the first case where there is a single customer, Feng et al. (2016) propose an online algorithm with a competitive ratio of 3, and for the second case where there are multiple customers and all the delivery vehicles have the same capacity, they propose a similar algorithm with a competitive ratio of $2\min\{c, 1+(1-1/c)(\Sigma f_k)/f_{min}\}$, where c is the vehicle capacity, f_{min} is the minimum transportation cost, and Σf_k is the sum of transportation costs to different customers.

For the online parallel-machine scheduling problem with the objective of minimizing the maximum completion time of the jobs, which can be viewed as a special case of our Problem P2 without the delivery part, Chen and Vestjens (1997) show that assigning available jobs to available machines by the longest processing time (LPT) first rule gives a 1.5-competitive ratio. For a single machine problem where each job is delivered individually to its destination after processing and the objective is to minimize $D_{\max}$,

Hoogeveen and Vestjens (2000) present an optimal 1.618-competitive algorithm. Several variants of this problem are also considered in the literature (e.g., Liu et al. 2010, Tian et al. 2012). For the same problem but with parallel machines, Hall and Shmoys (1989) design a 2-competitive on-line algorithm. There are also studies that consider similar problems where jobs are delivered in batches to a single destination. Ng and Lu (2012) consider a single-machine problem with a single delivery vehicle and the objective of minimizing $D_{\max}$. More recently, Han et al. (2015) study a variety of single-machine and parallel-machine problems with a single customer with the objective of minimizing the sum of $D_{\max}$ and the total transportation cost.

References Cited:

Chen, B., A.P.A. Vestjens. 1997. Scheduling on identical machines: How good is LPT in an on-line setting? *Operations Research Letters*. 21: 165-169.

Feng, X., Y. Cheng, F. Zheng, Y. Xu. 2016. Online integrated production-distribution scheduling problems without preemption. *Journal of Combinatorial Optimization*. 31(4): 1569-1585.

Hall, L.A., D.B. Shmoys. 1989. Approximation schemes for constrained scheduling problems. 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 134–139.

Han, B., W. Zhang, X. Lu, Y. Lin. 2015. On-line supply chain scheduling for single-machine and parallel-machine configurations with a single customer: Minimizing the makespan and delivery cost. *European Journal of Operational Research*. 244: 704-714.

Hoogeveen, J.A., A.P.A. Vestjens. 1996. Optimal online algorithms for single-machine scheduling. in Proceeding of the 5th International Conference on Integer Programming and Combinatorial Optimization. Lecture Notes in Computer Science, 1084: 404-414.

Hoogeveen, J.A., A.P.A. Vestjens. 2000. A best possible deterministic on-line algorithm for minimizing maximum delivery time on a single machine. *SIAM Journal on Discrete Mathematics*. 13: 56-63.

Liu, C.-H. 2011. Using genetic algorithms for the coordinated scheduling problem of a batching machine and two-stage transportation. *Applied Mathematics and Computation*. 217(24): 10095-10104.

Liu, M., C. Chu, Y. Xu, F. Zheng. 2010. An optimal online algorithm for single machine scheduling with bounded delivery times. *European Journal of Operational Research*. 201: 693-700.

Liu, P., X. Lu. 2009. On-line scheduling of parallel machines to minimize total completion times. *Computers & Operations Research*. 36: 2647-2652.

Lu, X., R.A. Sitters, L. Stougie. 2003. A class of on-line scheduling algorithms to minimize total completion time. *Operations Research Letters*. 31(3): 232-236.

Ng, C.T., L. Lu. 2012. On-line integrated production and outbound distribution scheduling to minimize the maximum delivery completion time. *Journal of Scheduling*. 15: 391-398.

Sitters, R. 2010. Efficient algorithms for average completion time scheduling. in Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization. Lecture Notes in Computer Science, 6080: 411-423.

Tian, J., T.C.E. Cheng, C.T. Ng, J. Yuan. 2012. An improved on-line algorithm for single parallel-batch machine scheduling with delivery times. *Discrete Applied Mathematics*. 160: 1191-1210.

MISTA 2019

Online Integrated Optimization Model for Container Rehandling and Pickup Sequence in Container Terminals

Su-min Chen • Qing-cheng Zeng

1 Introduction

During containers retrieving process, the unproductive rehandling moves occur when containers stacked above the target container for retrieval, which decreases the efficiency of the terminal operation, increases the delay of external trucks, and decreases vessel operation efficiency. The container relocation problem(CRP) is a classic combinatorial optimisation problem, which is to retrieve all the containers (outbound or inbound) from a bay in the prescribed order while minimizing the number of rehandles. Different from the CRP of outbound containers, the exact retrieval sequence of inbound containers in a bay is unknown in advance due to the uncertainty of truck retrieval time. With the development of GPS (Global Positioning System) and IoT (Internet of things) and the implementation of truck appointment systems, truck arrival time can be revealed during the retrieval process. This paper introduces and investigates such online version of the problem with the real-time truck arrival information. We also study a class of flexible service policies, which considers adjustments to the container pickup sequence within each group to minimize the total number of rehandles. In summary, this problem is an online integrated problem to optimize the pickup sequence and the container rehandling strategy simultaneously.

Due to the problem class being NP-hard and time limit of online optimization, we decompose this problem into two subproblems. One is offline Container relocation problem which is to get the set of optimal rehandling plans of all possible pickup sequences before retrieval operations begin. With the output of previous problem and real-time truck arrival information, another is online integrated problem which is to choose the optimal pickup sequence and its corresponding rehandling plan. In order to address subproblems, we propose rehandling model with time windows and online integrated optimization model respectively. Finally, we design an algorithm based on decision tree search algorithm and greedy strategy to solve the model and analyze the validity of our model and algorithm.

Numerical experiments indicate that indicate that the proposed algorithms can effectively solve the addressed problem. In most cases, optimizing the rehandling scheme and pickup sequence simultaneously can greatly reduce the number of rehandles, but may increase the average waiting time of external trucks. Moreover, our model controls wait time of each external truck, thus can contribute to the stability of the service level in container terminals. The proposed method provides an efficient tool for container terminals to reduce the number of rehandles and obtain relocation patterns in real time.

Su-min Chen
Department of Maritime Economics and Management, Dalian Maritime University, Dalian
E-mail: 1441794518@qq.com

Qing-cheng Zeng
Department of Maritime Economics and Management, Dalian Maritime University, Dalian
E-mail: qzeng@dlmu.edu.cn

MISTA 2019

Two-machine Openshop scheduling with Two Agents

Peihai Liu · Xiwen Lu

1 Introduction

We consider a two agent scheduling problem, in which each agent with a set of non-preemptive jobs, compete to perform their respective jobs on a common processing resource. Since two-agent scheduling problem was first studied by Baker and Smith[4] and Agnetis et al. [1], hundreds of follow-up papers have been published on it [2,3,5–8] , and the following three models of two-agent scheduling problem are wisely studied, in which f and g respectively denote the objective functions of the two agents A and B.

- *Constrained optimization*: Let Q be an upper bound of the function g. The goal of the problem is to find a schedule σ such that the objective function f is minimized under the constraint that $g \leq Q$.
- *Combination optimization*: Given two positive number λ_1 and λ_2, the goal of the problem is to find a schedule σ so that the value of the combination $\lambda_1 \cdot f + \lambda_2 \cdot g$ is minimized.
- *Pareto optimization*: The goal of this model is to enumerate all Pareto optimal points and, for each Pareto optimal point, find the corresponding Pareto optimal schedule.

In this paper, we focus a constrained optimization in a two-machine openshop system. There are two agents A and B, each of which has a fixed set of nonpreemptive jobs to be processed on the same two-machine open shop system. The two machine are denoted by M_1 and M_2, respectively. Without causing any confusion, we also use A and B to denote the job sets belonging to agent A and agent B, respectively, where A consists of n_A jobs and B has n_B jobs, where $A \bigcap B = \emptyset$ and $n_A + n_B = n$. We use J_j^X to denote the job number j in subset $X, X = A$ or B. Each job J_j^X consists of 2

Peihai Liu
Department of Mathematics, East China University of Science and Technology, Shanghai 200237, China.
E-mail: pliu@ecust.edu.cn

Xiwen Lu
Department of Mathematics, East China University of Science and Technology, Shanghai 200237, China.
E-mail: xwlu@ecust.edu.cn

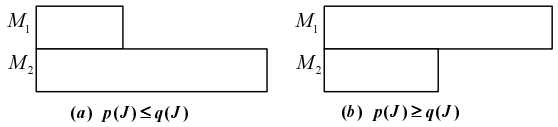


Fig. 1 Illustration of the structure of schedule for $O2||C_{max}$: Type I

operations $O_{ij}^X (i = 1, 2)$ where O_{ij}^X must be processed on machine M_i. For job J_j^X, we use p_j^X, q_j^X to denote the processing times of operation O_{1j}^X and O_{2j}^X, respectively. Let C_{max}^X denote the makespan of agent $X = A, B$. Given job sets A and B, and a factor $Q > 0$, the goal is to find a schedule minimizing C_{max}^A subject to $C_{max}^B \le Q$. This problem is denoted as $O2||C_{max}^A : C_{max}^B \le Q$.

We use $J = A \bigcup B$ to denote the job set of a given instance I. Given a subset $S \subseteq J$, we denote by $p(S)(q(S))$ the total processing time on $M_1(M_2)$ of jobs belonging to S. We also use $p_{min}(S)(q_{min}(S))$ the minimal processing time on $M_1(M_2)$ of jobs belonging to S.

2 Main Results

2.1 Non-approximability

We will show that if there is an approximation algorithm with a worst-case ratio of r for any $r \ge 1$, then it can be used to establish a polynomial time algorithm for solving the PARTITION problem, which is NP-hard, leading to a contradiction (if P $\neq$ NP). Hence, such an algorithm for the considered problem cannot exist unless $P = NP$.

Lemma 1 *Given an positive number $r \ge 1$, if there exists a polynomial time approximation algorithm A with a worst-case ratio of r for $O2||C_{\max}^A : \theta C_{\max}^B < Q$, then there exists a polynomial time algorithm for the PARTITION problem.*

2.2 $O2||C_{max}$

We first study the structure of schedules to $O2||C_{\max}$. Suppose all jobs are sequenced in nondecreasing order of the value $p_i + q_i$, i.e., $p_1 + q_1 \ge p_2 + q_2 \ge \ldots, \ge q_n + q_n$. Given a job set S and a job $j \in S$, we say J_j is large if $p_j + q_j > p(S)$ or $p_j + q_j > q(S)$. For a job $j \in S$, we say J_j is small if $p_j + q_j \le p(S)$ and $p_j + q_j \le q(S)$. For convenience, we define $X(Y)$ as the set of the large(small) jobs in S.

Lemma 2 *Given a job set S for problem $O2||C_{max}$, if $p(S) \le q(S), p_{min}(X) \le q(Y)$ (or $p(S) \ge q(S), q_{min}(X) \le p(Y)$), then there exists an schedule for job set S with the structure illustrated in Fig. 1, in which all jobs are continuously processed on each machine.*

If a set job S satisfies the conditions in Lemma 1, we say S belongs to Type I. If $p_1 + q_1 > p(S)$ and $p_1 + q_1 > q(S)$, we say S belongs to Type II. otherwise, S belongs to Type III

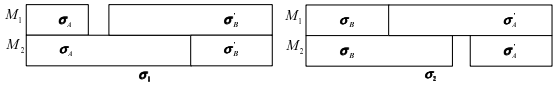


Fig. 2 Illustration of Algorithm A1

2.3 Polynomially solvable cases

Algorithm A1

Step 1. If job set A and B are belong to Type I, construct two schedule with the structure illustrated in Fig. 1 for A and B, respectively. Denote by σ_A and σ_B.

Step 2. Denote by σ'_A and σ'_B by horizontally reversing σ_A and σ_B, respectively.

Step 3. Merge σ_A and σ'_B and denote it by σ_1. Merge σ_B and σ'_A and denote it by σ_2(See Fig 2).

Step 4. If σ_1 does not violate the constrain, choose σ_1 as the solution; otherwise choose σ_2 as the solution.

Lemma 3 *If job set A and B are belong to Type I, Algorithm A1 give the optimal schedule in $o(n)$ time.*

2.4 A pseudo-polynomial-time algorithm and FPTAS

A pseudo-polynomial time dynamic programming algorithm is presented, which means that $O2||C^A_{max} : C^B_{max} \leq Q$ is NP-hard in the ordinary sense. The pseudo-polynomial time dynamic programming algorithm takes time in $O(n^5T^6)$, where $T = \max\{p(J), q(J)\}$.

By Lemma 1, there is no polynomial time approximation algorithm with an constant worst case ratio for $O2||C^A_{max} : C^B_{max} \leq Q$. Thus $O2||C^A_{max} : C^B_{max} \leq Q$ does not admit a fully polynomial time approximation scheme. We study a relaxation version, in which the constrain can be violated by a factor of ε. With violating the constraint a factor of ε, we provide a FPTAS with time complexity $O(\frac{n^5}{\varepsilon^6})$ for $O2||C^A_{max} : C^B_{max} \leq Q$.

References

1. Agnetis, A., Mirchandani, P.B., Pacciarelli, D., Pacifici, A., Scheduling problems with two competing agents, Operations Research, 52, 229-242 (2004)
2. Agnetis, A., Pacciarelli, D., Pacifici, A., Multi-agent single machine scheduling, Annals of Operations Research, 150, 3-15 (2007)
3. Agnetis, A., Pascale, G., Pcciarelli, D., A Lagrangian approach to single-machine scheduling problems with two competing agents, Journal of Scheduling, 12, 401-415(2009)
4. Baker, K.R., Smith, J.C., A multiple-criterion model for machine scheduling, Journal of Scheduling, 6, 7-16(2003)
5. Fan, B.Q., Cheng, T.C.E., Two-agent scheduling in a flowshop, European Journal of Operational Research, 252: 376-384(2016)
6. Leung, J.Y.T., Pinedo, M., Wan, G., Competitive two-agent scheduling and its applications, Operations Research, 58(2), 458-469(2010)
7. Luo, W., Chen, L., Zhang, G., Approximation schemes for two-machine flow shop scheduling with two agents, Journal of Combinatorial Optimization, 24, 229-230(2012)
8. Perez-Gonzalez, P., Framinan, J.M., A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: multi-agent scheduling problems, European Journal of Operational Research, 235(1), 1-16(2014)

MISTA 2019

An enhanced branch-and-price algorithm for the integrated production scheduling and delivery routing problem

Peiyang He • Kunpeng Li

The efficient coordination of intelligent manufacturing and JIT delivery has received much attention from intelligent manufacturing enterprises. This paper studies an integrated production and transportation problem and takes both the service-related criterions and cost-related criterions into the objective. There are a set of customers located at different locations, a single machine in the plant and limit but sufficient vehicles in the system. Each of the customers has an order with different job numbers. The machine can produce jobs continuously with constant production rate. Limit but sufficient numbers of homogeneous vehicles with limited capacity are available to deliver finished products to customers. Based on the characteristics of the problem, this paper proposes a new set-covering formulation and the properties of the optimal solution. An improved branch-and-price algorithm is designed to solve the problem. Computational results show that the proposed formulation and algorithm can offer optimal solutions in reasonable time.

Peiyang He
School of management, Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: 929939474@qq.com

Kunpeng Li
School of management, Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: likp@hust.edu.cn

MISTA 2019

Two single-machine scheduling problems with rejection

Ying Li[a], **Lingfa Lu**[a,*], *and* **Liqi Zhang**[b]

[a] School of Mathematics and Statistics, Zhengzhou University,
Zhengzhou, Henan 450001, P.R. China

[b] College of Information and Management Science, Henan Agricultural University,
Zhengzhou, Henan 450003, P.R. China

Abstract. In this paper, we consider two single-machine scheduling problems with rejection. There are n jobs $J_1, J_2, \cdots, J_n$ and a single machine. Each job has a processing time p_j, a weight w_j and a rejection cost e_j. Job J_j is either accepted and then processed on the single machine, or is rejected by paying the corresponding rejection cost e_j. Let A and R be the set of the accepted jobs and the set of the rejected jobs, respectively. Furthermore, for each job $J_j \in A$, let C_j be the completion time of J_j. We define $C^k_{\max} = \{C^k_j : J_j \in A\}$ and $WC_{\max} = \{w_j C_j : J_j \in A\}$ as the k-th power of the makespan and the weighted makespan of the accepted jobs, respectively. The objective is to minimize the sum of the k-th power of the makespan (or the weighted makespan) of accepted jobs and the total rejection cost of rejected jobs. Using the general notation for scheduling problems, our problems can be denoted by $1||C^k_{\max} + \sum_{J_j \in R} e_j$ and $1||WC_{\max} + \sum_{J_j \in R} e_j$.

Firstly, we show that both of two problems are binary NP-hard. Secondly, we propose a pseudo-polynomial dynamic programming algorithm and some effective approximation algorithms for each problem. Finally, based on the dynamic programming algorithm and the data rounding or vector trimming technique, we also obtain a fully polynomial-time approximation scheme (FPTAS) for each problem.

Key words: Scheduling with rejection, NP-hard, dynamic programming, approximation algorithms, FPTAS

*Corresponding author: Lingfa Lu. Email address: lulingfa@zzu.edu.cn

MISTA 2019

A Modified Teaching-Learning-Based Optimization Algorithm for Distributed Job Shop Scheduling Problem with fuzzy processing time

Jin Xie, Liang Gao, Xinyu Li*

Abstract Considering the imprecise or fuzzy nature of the data in the real-world problem, this paper deals with the distributed shop scheduling problem (DJSSP) with fuzzy processing time. DJSSP is a generalization of the classic job shop scheduling problem (JSSP), characterized by simultaneously assigning jobs to different factories/workshops and determining their schedules. The uncertain processing time is represented as a triangular fuzzy number. To deal with this problem, a mixed integer programming model is formulated, and a modified teaching-learning-based optimization (MTLBO) is proposed to minimize the maximum fuzzy completion time. In the MTLBO, a new self-learning mechanism is introduced to enable students not only to be taught by a teacher and classmates, but also to learn by themselves. Some numerical experiments are designed to test the performance of the MTLBO, and compare it with other algorithms from literatures. The experimental results demonstrate that the MTLBO can solve the DJSSP with fuzzy processing time effectively.

Keywords Distributed Job Shop Scheduling Problem; Fuzzy processing time; Triangular fuzzy number; Teaching-learning-based optimization

Jin Xie
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
E-mail: 233773979@qq.com

Liang Gao
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
E-mail: gaoliang@mail.hust.edu.cn

Xinyu Li
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
E-mail: lixinyu@mail.hust.edu.cn

MISTA 2019

A Discrete Artificial Bee Colony for Heterogeneous Distributed Hybrid Flowshop Rescheduling Problem

Yingli Li, Xinyu Li, Liang Gao*

Abstract The hybrid flowshop scheduling problem has important applications in modern industries. Meanwhile, realistic scheduling often encounters several types of dynamic events. Under the distributed and dynamic environment, this paper addresses the heterogeneous distributed hybrid flowshop rescheduling with a discrete artificial bee colony (DABC) algorithm to minimize the maximum completion time. Different from the existing distributed researches, this study considers the heterogeneous factory environment to satisfy the real-life production situation. A batch of unrelated jobs (orders) can be processed in a set of factories with the same model of hybrid flowshop in different locations. Meanwhile, these factories exist different structures — different numbers of stages with unrelated parallel machines. In the proposed DABC, the novel two-level encoding is used to represent the candidate solutions. And, the effective machine selection rule is used to decode the high quality solutions. The neighborhood search based on the critical factory is proposed and the hybrid operator is adopted to generate neighboring solutions in the search process. New test instances are designed and compared the proposed DABC with other algorithms from literatures. The experimental results demonstrate the effectiveness of the proposed DABC algorithm.

Keywords Distributed Hybrid FlowShop Scheduling Problem; Heterogeneous Distributed Scheduling;Dynamic Scheduling; Completion Time; Artificial Bee colony

Yingli Li
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
E-mail: lyl_elena@163.com

Xinyu Li
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
E-mail: lixinyu@mail.hust.edu.cn

Liang Gao
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
E-mail: gaoliang@mail.hust.edu.cn

MISTA 2019

Energy-efficient distributed flow shop scheduling using a modified MOEA/D

Guangchen Wang • Xinyu Li • Liang Gao* • Peigen Li

Abstract Welding is one of the most important manufacturing processes and widely exists in the modern manufacturing industry, which produces enormous energy consumption. Therefore, it is of great significance to take energy consumption into account in welding shop scheduling problem. To reduce the delivery and manufacturing risk and cost, distributed welding flow shop contains several welding flow shops as a new trend in the real-world manufacturing environment. This study is aimed at solving distributed welding flow shop scheduling problem (DWFSP) under the consideration of energy efficiency and productivity. And a multi-objective mixed integer programming model is presented based on the sub-problem with allocating jobs among factories, scheduling the jobs in each factory and determining the amount of multi-machines upon each job. A modified MOEA/D is proposed to optimize the total energy consumption and makespan simultaneously. The real-world case study for distributed welding flow shop is presented. The results show the effectiveness of the proposed method.

MISTA 2019

How to schedule jobs with limited number of calibrations

Vincent Chau · Shengzhong Feng · Minming Li · Yinling Wang · Guochuan Zhang · Yong Zhang

1 Introduction

The scheduling problem of minimizing the number of calibrations has been introduced by Bender et al. [1]. It is motivated by the Integrated Stockpile Evaluation (ISE) program at Sandia National Laboratories for testing in contexts where safety mistakes may have serious consequences. More generally, this problem concerns the manufacture of modern industrial products according to their exact standards like digital cameras or processors. The machines that make these products are strict and execute dedicated tasks. Therefore, they need to be calibrated carefully before they can perform any job. The products can be considered reliable only if the machines have been well calibrated beforehand. These calibrations can be very expensive and can be much more expensive than the running cost of the machines.

The initial scheduling problem is as follows. We are given a set of n jobs that need to be scheduled on a set of m identical machines. Each job j has a release time r_j, a deadline d_j and a processing time p_j. A job is scheduled if it is processed entirely inside its interval $[r_j, d_j)$. However, we can schedule jobs only if we perform calibrations beforehand. A calibration activates the machine instantaneously for a period of T time units, and the machine can start to process jobs as soon as it is calibrated. After T time units, the machine cannot schedule any jobs unless we perform another calibration. The goal is to find a feasible schedule such that all jobs are scheduled with the minimum number of calibrations.

Vincent Chau, Yong Zhang
Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China

Shengzhong Feng
National Supercomputing Centre in Shenzhen, China

Minming Li
City University of Hong Kong, Hong Kong, China

Yinling Wang
Dalian University of Technology, Dalian, China

Guochuan Zhang
Zhejiang University, Hangzhou, China

Related Work. Bender et al. [1] considered the case of unit-processing time and proposed an optimal algorithm for this problem when there is only one available machine, and a 2-approximation algorithm when there are multiple machines. Although this problem can be solved in polynomial time for the single machine case, it remains open whether it is polynomial or NP-hard for the multiple machines case.

Fineman and Sheridan [2] generalized the calibration model and considered the case where jobs do not have unit processing time anymore and preemption of jobs is not allowed. They gave an algorithm which is given extra resources over the optimal solution. In other words, the algorithm is allowed to use αm-machines (instead of m machines) for some $\alpha > 1$ and the processing time of a jobs j is p_j/s (instead of p_j).

Our problem. In this paper, we study the *throughput* version of the scheduling problem in the seminal paper [1]. We are given a set of n jobs with their respective release time, deadline and processing time. We have a limited budget K in the number of calibrations and each job is associated with a weight. The goal is to find a subset of jobs that can be scheduled with at most K calibrations on m indentical machines such that the total weights of the selected jobs is maximized. Note that the latter part can be solved with the algorithm proposed by Baptiste et al. [4].

Our contribution. When jobs have unit processing time, we present a $\frac{1}{3}$-approximation algorithm. Furthermore, we consider the generalization where jobs have arbitrary processing time, preemption of jobs is allowed but each job must be scheduled within the same calibration. The problem is weakly NP-hard and we give a polynomial time algorithm whose approximation ratio is $\frac{1}{18}(1-\varepsilon)$ for $\varepsilon > 0$.

2 The algorithm

The idea of the algorithm is as follows: As long as the number of used calibrations is less than the given budget, we choose the jobs that can be scheduled within a single calibration such that the total profit is maximized. Instead of considering every subset of jobs that can be scheduled within a calibration, we show that it is sufficient to consider a limited number of starting times of a calibration. Once we set the starting time t of the calibration, we aim to find the set of jobs that can be scheduled in the interval $[t, t+T)$ whose profit is maximum. Such a solution dominates any other solutions with the same starting time of calibration, because in our algorithm we select the set of jobs with maximum profit. Finally, the number of different starting times of a calibration can be bounded by $O(n^2)$.

Proposition 1 ([3]) *There exists an optimal solution in which each calibration starts at a time in* $\{d_i - n, d_i - n + 1, \ldots, d_i \mid i = 1, \ldots, n\}$.

Theorem 1 *Algorithm 1 is a* $\frac{1}{3}$*-approximation for the problem of maximizing the weighted throughput of unit-processing time jobs with at most* K *calibrations and whose running time is* $O(Kn^6)$.

A naive way to extend to the arbitrary processing time case is to create as many unit-time jobs as the processing time for each job. However, this approach leads to a pseudo-polynomial time algorithm. We show that by considering fewer possibilities on the starting time of the calibrations, we lose a factor of 6 on the profit of the scheduled jobs.

Algorithm 1 Greedy algorithm for the problem of maximizing the total weights of selected jobs with a calibration budget

Require: K, $J = \{1, 2, \ldots, n\}$
1: $\Psi \leftarrow \bigcup_i \{d_i - n, d_i - n + 1, \ldots, d_i\}$.
2: $C \leftarrow \emptyset$ //set of calibrations
3: **while** $|C| \leq K$ **do**
4: $S \leftarrow \emptyset$ // set of choices of calibrations
5: **for** $t \in \Psi$ such that at most $m - 1$ calibrations in interval $[t, t + T]$ **do**
6: $S_t \leftarrow$ Solve the maximum weighted throughput scheduling problem in interval $[t, t+T]$ with the set of jobs J using the algorithm proposed by Baptiste et al. [4]
7: $S \leftarrow S \cup \{S_t\}$
8: **end for**
9: $S' \leftarrow \arg\max_{S_t \in S}\{S_t\}$
10: $C \leftarrow C \cup S'$
11: $J \leftarrow J \setminus S'$
12: **end while**
13: **return** C

Proposition 2 *There exists a $\frac{1}{6}$-approximation schedule such that each calibration starts or ends at a time in $\{d_j + aT \mid a = -n, -n + 1, \ldots, n - 1, n$ and $j = 1, \ldots, n\}$.*

Theorem 2 *When jobs have arbitrary processing times and arbitrary weights, Algorithm 1 runs in time $O(Kn^7/\varepsilon^2)$ and has an approximation ratio of $\frac{1}{18}(1 - \varepsilon)$.*

When computing the maximum weighted throughput for jobs with arbitrary processing time, we use an FPTAS derived from the algorithm in [5].

3 Conclusion

This work is the first to our knowledge to study throughput maximization scheduling problem with calibrations and the first to give polynomial-time algorithms with constant approximation ratio. When jobs have arbitrary processing times, we get a $\frac{1}{18}(1 - \varepsilon)$-approximation polynomial time algorithm. On the other hand, we have a $\frac{1}{3}$-approximation polynomial time algorithm for the unit processing time case. A natural question is whether it is possible to improve the approximation ratio.

References

1. Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, Cynthia A. Phillips (2013). *Efficient scheduling to minimize calibrations*. SPAA 2013: 280-287
2. Jeremy T. Fineman, Brendan Sheridan (2015). *Scheduling Non-Unit Jobs to Minimize Calibrations*. SPAA 2015: 161-170
3. Eric Angel, Evripidis Bampis, Vincent Chau, Vassilis Zissimopoulos (2017). *On the Complexity of Minimizing the Total Calibration Cost*. FAW 2017: 1-12
4. Philippe Baptiste, Marek Chrobak, Christoph Dürr, Wojciech Jawor, Nodari Vakhania (2004). *Preemptive scheduling of equal-length jobs to maximize weighted throughput*. Oper. Res. Lett. 32(3): 258-264
5. Eugene L. Lawler (1990). *A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs*. Annals of Operations Research 26(1): 125-133

MISTA 2019

Combinatorial Benders approach to solve the Quay Crane Scheduling Problem

Defeng Sun • Lixin Tang

Abstract We propose an exact decomposition approach to solve the Quay Crane Scheduling Problem (QCSP) in container terminals. The problem is decomposed into two easier problems, i.e. a routing master problem and a scheduling sub-problem. The master problem probes possible routing solutions while the sub-problem verifies the feasibility of the routing solution and determines the completion times of each task and each quay crane. Several types of valid inequalities and combinatorial cuts are generated and added into the master problem to reduce the probing space and further accelerate the solution approach. Computational tests on the benchmark instances verify the effectiveness of the proposed approach.

1 Introduction

In maritime transportation, the use of containers improved shipping operations and induced strong growth of the world economy and globalization. Nowadays, the growth of ultra large container vessels has placed increasing challenges in managerial practices at container terminals, which strive to improve their services and meet demands to attract a larger magnitude of shipments. The need for more effective management of logistics activities at modern container terminals is well recognized.

Quay cranes (QCs) are the crucial equipment used in container terminal to load and unload the containers onto and from the vessels. The performance of a terminal is often measured in terms of the stay times of container vessels, which is highly related to the seaside operations planning. The seaside operations mainly contain berth allocation, quay crane assignment and quay crane scheduling. We refer the reader to the reviews in Bierwirth and Meisel (2010, 2015).

The objective of the quay crane scheduling problem (QCSP) is to determine a sequence of unloading and loading movements for cranes assigned to a vessel in order to minimize the vessel completion time as well as the crane idle times, according to Moccia et. al. (2006). The idle times here are mainly caused by the spatial requirements of non-crossing and safety-distance-maintaining among the QCs, because that the QCs are mounted on the same rail track.

This paper tackles the QCSP, specially, the class of <Group,prec|move|cross,save| max(compl)> according to classification scheme provided by Bierwirth and Meisel (2010, 2015), which considers QC interference, non-preemption, bidirectional movement and with the objective of minimizing the latest completion time. A task consists of the loading or unloading of a single or a group of containers in certain bays of the vessel, and container groups in the same bay can be assigned to different QCs. Also, there exist precedence relationships among

Defeng Sun
Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern Univ.), Ministry of Education, China
E-mail: sundefeng@ise.neu.edu.cn

Lixin Tang
Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern Univ.), Ministry of Education, China
E-mail: lixintang@ise.neu.edu.cn

different tasks in the same bay, considering the stacking positions of the containers as shown in Fig 1.

This paper follows the research line of Sampaio et al. (2016) to develop exact decomposition methods for QCSP. The contribution of this work is highlighted as follows:.

1) a tight lower bound strategy which could probably obtain the optimal solution in advance;

2) various valid inequalities which enable the routing master problem to obtain better lower bound and reduce the generation of solutions with subtours and QC interference;

3) several types of combinatorial cuts to accelerate the convergence of the solution approach.

4) Computational tests on benchmark data shows that the proposed approach is able to solve most of the large-scale instances which can be tackled by heuristics only in previous lituerature.

References

1. Bierwirth, C. and Meisel, F. A survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research, 202(3), 615–627. (2010)
2. Bierwirth, C. and Meisel, F. A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research, 244, 675–689. (2015)
3. Moccia, L., Cordeau, J. F., Gaudioso, M., & Laporte, G. A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. Naval Research Logistics, 53(1), 45–59. (2006).
4. Sampaio, A., Urrutia, & Oppen, J. A Decomposition Approach to Solve The Quay Crane Scheduling Problem. (2016)

MISTA 2019

Scheduling for refinery processes with consideration of uncertain demand

Lijie Su • Yiping Huang • Lixin Tang

Refinery production includes a series of continuous and complicated operation processes, which transform different types of crude oil into several standard types of oil product. The demand for oil products from the international and national markets often varies, which mostly determines the profit of the refineries. This work addresses the scheduling for refinery processes with consideration of the uncertain demands of oil products. The stochastic scheduling problem of refinery processes under uncertainty is defined. The uncertainty of demand is represented into a scenario tree. We propose an Mixed Integer Nonlinear Programming (MINLP) model for the scheduling problem based on the continuous-time representation. Aiming at the proposed MINLP model based on a scenario tree, we develop an Outer Approximation solution method with the Fix-and-Relax strategy to solve the large-scale cases. Computational results demonstrate the robustness of the proposed stochastic model and the efficiency of the developed solution method compared with standard MINLP solvers.

Lijie Su
Institute of Industrial and Systems Engineering, Northeastern University, Shenyang, China
E-mail: sulijie@ise.neu.edu.cn

Yiping Huang
Institute of Industrial and Systems Engineering, Northeastern University, Shenyang, China

Lixin Tang
Institute of Industrial and Systems Engineering, Northeastern University, Shenyang, China

MISTA 2019

Coordinated scheduling of transportation and production on single batch machine based on reinforcement learning

Gong Hua • Tang Lixin

Abstract Motivated by the ingot process in the iron and steel industry, a coordinated scheduling problem with transportation and batching production is considered. The jobs with dynamic release times are first transported by multi-vehicles to further production area. Then they are processed in batches on a single batching machine. The batching machine can process several jobs simultaneously as a batch. The processing time of a batch is the longest processing time of the jobs among this batch. The objective is to minimize the total completion time. A mixed integer programming model is formulated to describe the problem proposed in this paper. We transfer our coordinated scheduling problem proposed in this paper to a multi-stage sequential decision-making problem to develop reinforcement learning algorithm. The real-time state features of the scheduling problem with transportation and production in the system are set as input, and the heuristic algorithms including scheduling rules are combined as actions. We propose a Q-Learning algorithm based on reinforcement learning to solve linear value function approximation for the problem. The computational results show the feasibility and effectiveness of the proposed reinforcement learning algorithm.

First Author: Gong Hua
Affiliation: Shenyang Ligong University
E-mail: gonghua@sylu.edu.cn

Second Author: Tang Lixin
Affiliation: Key Laboratory of Data Analytics and Optimization for Smart Industry, Northeaster University
E-mail: lixintang@ise.neu.edu.cn

MISTA 2019

Integration Transportation Problem in the Open-Pit Mine via Decomposition based Reinforcement Learning

Fengyuan Shi • Lixin Tang

Abstract This paper focus on transportation problem in the open-pit mine, which consist of vehicle config problem, path planning problem and vehicle scheduling problem. This paper proposed a integration framework: bilevel markov decision process to deal with the whole transportation problem in the open-pit mine. After proof convergence and optimality of open-pit mine transportation bilevel markov decision problem formulation under value iteration and policy iteration algorithm, then solve the bilevel markov decision process formulation using decomposition based reinforcement learning proposed by this paper. Numerical experiments verify the effectiveness of the proposed method.

Fengyuan Shi
Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern Univ.), Ministry of Education, China
E-mail:im.fengyuan.shi@outlook.com

Lixin Tang
Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern Univ.), Ministry of Education, China
E-mail: tanglixin@ise.neu.edu.cn

MISTA 2019

Feasibility of Digital Twin on Digital Port Designing

Ang Yang • Qingcheng Zeng

Abstract
Seaport is used as a transportation hub for responding to the critical tasks, involving maritime logistic, vessel berthed, and packaging cargos. Currently, the advantage of seaport type is an automated terminal. During the development of information technology, the automated port is going to become a digital/smart port. To design the digital port, it should consider two aspects – industry 4.0 and digital twin. From the physical product, industry 4.0 is the key to support the digital port designing. Meanwhile, the information technologies are critical part of achieving the demands of smart port because all the physical products should be connected via information or IoT system. Digital twin, which is the next-generation system, is recognised as the solution of smart designing. As a consequence, the digital twin should be prioritised to designing strategy of digital port. Digital port is the next-generation port. It leads that many dimensions of digital port designing are unclear in research and industry domain. To achieve successfully designing a digital port, the first consideration is about the standard of digital port. Moreover, some critical research issues should be discussed like how to build a digital twin of digital port; What kind of digital twin is suit for digital port designing; and what the digital twin could bring to operation of digital port. The qualitative research method is adopted, including the case study and cross-case analysis to verify the feasibility associated with digital twin running on digital port designing. The successful factors collected from other smart domains and characteristics of seaport designing should be cooperated considered for delivering the construction scheme in relation to digital port. Therefore, the discussion associated with feasibility is vital for the implementation of the digital port design.

Ang Yang
Shipping Economics and Management College, Dalian Maritime University
E-mail: ang.dlmu@outlook.com

Qingcheng Zeng
Shipping Economics and Management College, Dalian Maritime University
E-mail: zqcheng2000@hotmail.com

MISTA 2019

Big data analysis based robotic cellular manufacturing optimization model: Integrated production planning and preventive maintenance scheduling

Dan Luo •Zailin Guan

1 Introduction

The preventive maintenance and the production planning are usually considered separately. But preventive maintenance plans often conflict with production plans. Untimely preventive activity will not only disturb the production rhythm, but also interrupt the normal production. The preventive maintenance aims to carry out the equipment maintenance before the failure occurs, thus reducing the risk of fault occurrences on the production line, improving the utilization rate of machines, improving the reliability of the production line, and ultimately achieving the goal of improving the production capacity of production lines. However, the inappropriate preventive maintenance violates the original intention of preventive maintenance and even becomes an obstacle to the implementation of production plans [1-3]. Therefore, it has great significance to focus on the integration of the preventive maintenance plan and the production plan.

The product environments in precision manufactured systems are characterized by high-accuracy demands, strict quality requirements and complex production processes. Processes performed in these production centers require different types of chemicals under extreme conditions such as high or low pressure and temperature, and acidic environments in flicking heavy wear and tear on equipment. Therefore, preventive maintenance activities have attracted increased attention to avoid equipment failures. Preventive maintenance strategies are hence viewed as a mandatory tool to minimize breakdowns, excessive depreciation and total cost in precision manufactured industries. Moreover, the existing literatures didn't adequately capture the specifics of precision manufactured industries [4]. In this study, the preventive maintenance is studied for 3C（Computer, Communication, Customer） precision manufacturing industry, and the scheduling of production and preventive maintenance are also considered simultaneously.

Dan Luo
The State Key Laboratory of Digital Manufacturing Equipment & Technology,
Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: danluo@hust.edu.cn

Zailin Guan
The State Key Laboratory of Digital Manufacturing Equipment & Technology,
Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: zlguan@hust.edu.cn

2 Problem description

The flexible robotic manufacturing cell is mainly composed of the robot, the control system, the materiel input device (materiel storage), the manufacturing machine and the materiel output device [5]. The robot receives instructions from the control system and is responsible for some tasks such as the material handling and the assistant machining.

The structure of the flexible robotic manufacturing cell studied in this paper is shown in Fig. 1. Each robotic manufacturing cell consists of several computer numerical control (CNC) machines and one robot. A machining center consists of several robotic manufacturing cells, a materiel input device, a materiel output device, and a material conveyor. All robotic manufacturing cells share the same input and the output device in machining center. Materials are transferred from the input device to the robotic manufacturing cell through a material conveyor. The material enters the machining center through the input device, and then is transmitted to the designated CNC machine through the material conveyor. The robot is responsible for loading the material to the CNC machine. After the material is processed, the material is unloaded onto the material conveyor by the robot. Finally, the finished material leaves the machining center through the output device.

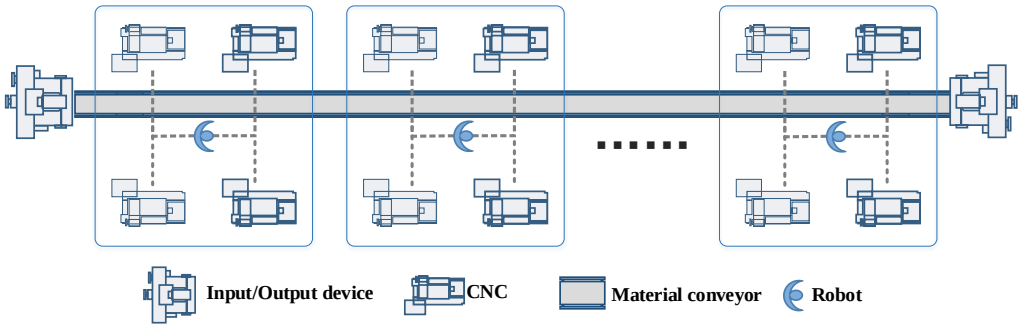


Fig.1. The structure of flexible robotic manufacturing cell

In this paper, the scheduling problem belongs to the flow shop problem and is described as follows. In a certain period of time, some arriving orders need to be allocated to robotic manufacturing cells by batch. When processing different types of part, it needs setup time to change the tool and the fixture. The preventive maintenance time is determined by the rules given by big data analysis. The order processing task, setup task, robot service task and preventive maintenance task are scheduled to minimize the completion time.

3 Big data preventive maintenance analysis based robotic cellular manufacturing scheduling optimization

Taking the robotic manufacturing cell as the research object, the prediction and analysis of preventive maintenance are carried out by using big data analysis technology. Furthermore, we can respond to changes of the order requirement and the prevent maintenance quickly, and optimize the scheduling of manufacturing cells. Finally, we can minimize the completion time and reduce the wastage of machine and robot under the condition of ensuring that the planning is completed on time.

3.1 LS-SVM regression algorithm based predictive model for preventive maintenance

Support Vector Machine (SVM) is a machine learning method. Least Squares Support Vector Machine (LS-SVM) is an improved algorithm of SVM. In current research, LS-SVM regression algorithm is improved. We establish an observation prediction model based on the combination of characteristic value and observation value, and propose a regression algorithm

of LS-SVM to train model. Through the improved algorithm, a prediction model is established to predict the tool wear of CNC machine, so as to determine the time of preventive maintenance. The improved LS-SVM regression algorithm is compared with the classical LS-SVM algorithm and BP neural network to analyze the performance of the prediction model. The overall idea is shown in Figure 2.

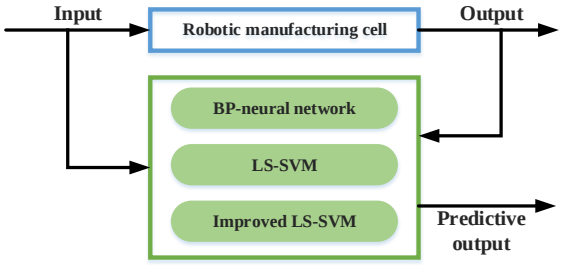


Fig.2. The predictive model for preventive maintenance

3.2 Integrated optimization model for robotic manufacturing cell planning and preventive maintenance

In current research, the goal of the integrated optimization model is to minimize the completion time. The scheduling rule of robot in robotic manufacturing cell is cyclic scheduling. There are many types of parts. And there is only one type of parts in each order. Changeover tasks are performed when their types are different. Preventive maintenance of the robotic manufacturing cell on CNC machine can be carried out when each order is completed and when changeover tasks are performing. After preventive maintenance, the CNC machine state becomes completely new, that is, the aging degree returns to zero. The latest start time of preventive maintenance is obtained by predictive model. If the CNC machine is not maintained before the latest start time of preventive maintenance, the CNC machine will broke down. The problem solved by this optimization model is how to arrange the order release sequence, processing time and preventive maintenance time to minimize the completion time under various constraints.

Acknowledgements This work was supported in part by the China Postdoctoral Science Foundation under Grant [2019M652665], and in part by the National Natural Science Foundation of China under Grant [51705379], Grant [51905196], Grant [71620107002], Grant [71131004], and Grant [71471071].

References

1. Al-Hinai N, Elmekkawy T Y. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm [J]. International Journal of Production Economics,2011, 132(2):279-291.[59]
2. Nouri H E, Driss O B, Ghédira K. Hybrid metaheuristics for scheduling of machines and transport robots in job shop environment[J]. Applied Intelligence, 2016, 45(3):1-21.
3. Mosley, S., T. Teyner, and R. M. Uzsoy. Maintenance Scheduling and Staffing Policies in a Wafer Fabrication Facility [J]. IEEE Transactions on Semiconductor anufacturing1998, 11(2): 316–323.
4. Tonke D, Grunow M. Maintenance, shutdown and production scheduling in semiconductor robotic cells [J]. International Journal of Production Research, 2018.
5. Yang Y, Chen Y, Long C. Flexible robotic manufacturing cell scheduling problem with multiple robots[J]. International Journal of Production Research, 2016, 54(22):6768-6781.

MISTA 2019

Operation scheduling of automated container terminals

Li Shan • Zeng Qingcheng

The operation process of the automated container terminal includes the loading and unloading operation, the horizontal transportation, the operation of the yard, and the entry and exit of the external container truck. In order to improve the production efficiency and service quality of the automated container terminal, the local and coordinated operation scheduling optimization of the above links is one of the hot issues in the port area. Automated container terminal can be regarded as a mixed flow workshop with complex process which includes berth allocation and quay crane scheduling, the distribution, routing and scheduling of AGVs, the scheduling of yard crane and the yard storage plan as well as predicting the arrival time of trucks, etc. In order to improve the production efficiency, it is necessary to overcome the dynamic instability factors in the operation of the terminal and optimize the overall operation process of the automated container terminal. Among them, the first three links are mainly based on mixed integer programming combined with various heuristic algorithms for scheduling optimization. In addition, the truck arrival prediction is solved by the support vector machine. The above methods can improve the production efficiency of the terminal to a certain extent, but based on some assumptions, it lacks the portrayal of the complex dynamic environment of the actual port. Thus, a better method is needed to solve the actual scheduling problem of the terminal. Machine learning has the characteristics of training its adaptability to complex dynamic environment through continuous learning, and combining heuristic algorithm can solve dynamic workshop scheduling problem. Therefore, it is significant to use machine learning combined with various heuristic algorithms to solve the optimization problem of automated container terminal.

Keywords: automated container terminal, scheduling and optimization, heuristic algorithm, machine learning

First Author
Li Shan
E-mail: shan.dlmu@outlook.com

Second Author
Zeng Qingcheng
E-mail: qzeng@dlmu.edu.cn

MISTA 2019

Solving the iron ladles allocation problem in molten iron transportation

Gongshu Wang • Zhenhao Yin • Lixin Tang

Abstract This paper studies the problem of allocating empty iron ladles to the blast furnaces for tapping operation arising in steel company. There are two types of iron ladles with different volumes, one called open-top ladle and another called torpedo ladle, available for molten iron transportation. In order to ensure the smoothness of tapping process, it is necessary to allocate a reasonable number of empty iron ladles to the taphole before each tapping preparation. Several particle factors that may affect the performance of tapping task should be taken into account when making empty iron ladle allocation plan. The first is the distance between the blast furnace and the location of empty iron ladle. The second is temperature state (hot, warm and cold) of each empty iron ladle. The last is that there is a fixed cost to pick up empty iron ladles from the same region by a locomotive. We formulate the problem as a mixed-integer programming (MIP) model and then reformulate it as a set partitioning type model using Dantzig-Wolfe decomposition. We propose a column generation-based algorithm to solve it. Computational experiments are carried out using a set of practical instances, and the results demonstrate the efficiency of the proposed algorithm.

Gongshu Wang
Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern Univ.), Ministry of Education, China.
E-mail: wanggongshu@ise.neu.edu.cn

Lixin Tang
Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern Univ.), Ministry of Education, China.
E-mail: lixintang@ise.neu.edu.cn

MISTA 2019

Two-agent scheduling on mixed batch machines to minimize the weighted sum of the agents' makespans

Guo-Qiang Fan[1,2] · **Jun-Qiang Wang**[1,2] · **Zhixin Liu**[3]

Abstract This research is motivated by the scheduling of vacuum heat treatment, which is widely used for the manufacturing of tools, molds and precision coupling parts. The processing time of a batch of workpieces is related to the maximum thickness and the total weight of the workpieces in the batch. In general, the weight of a workpiece is proportional to its thickness. Thus, the total processing time of a batch is modeled as the weighted sum of the maximum processing time and the total processing time of jobs in the batch. Wang et al. (2019) first proposed this kind of batch, and call it *mixed batch*, denoted by m-batch. This kind of batch is different from parallel batch (p-*batch*), serial batch (s-*batch*) and semi-continuous batch (c-*batch*). The main difference of these four types of batch model is how the processing time of a batch is determined. Following the research of Wang et al. (2019), this paper studies a two-agent scheduling problem on mixed batch machines. The objective is to minimize the weighted sum of the two agents' makespans. We show the *NP*-hardness of the problem, develop an approximation algorithm, and analyze its worst-case performance ratio.

Keywords Scheduling · Two-agent · Mixed Batch · Approximation Algorithm · Worst-case performance ratio

References

Wang JQ, Fan GQ, Liu Z (2019) Mixed batch scheduling on identical machines. Journal of Scheduling DOI 10.1007/s10951-019-00623-9

Guo-Qiang Fan
E-mail: pacpos.gqfan@gmail.com
✉ Jun-Qiang Wang
E-mail: wangjq@nwpu.edu.cn
Zhixin Liu
E-mail: zhixin@umich.edu
1 Performance Analysis Center of Production and Operations Systems (PacPos), Northwestern Polytechnical University, Xian, Shaanxi 710072, PR China
2 Department of Industrial Engineering, School of Mechanical Engineering, Northwestern Polytechnical University, Xian, Shaanxi 710072, PR China
3 Department of Management Studies, College of Business, University of Michigan-Dearborn, 19000 Hubbard Drive, Dearborn, Michigan 48126-2638, USA

MISTA 2019

Distribution route planning for station requirements material with split deliveries in assembly line

Weikang Fang • Zailin Guan • Lei Yue

1 Introduction

With the transformation of enterprise "manufacturing" to "intelligence", building a typical material distribution route planning model and designing intelligent algorithms could solve large-scale material distribution problems, for example genetic algorithm, ant colony algorithm, particle swarm algorithms and so on, which has become the key to intelligent handling, intensive intelligence and cost reduction.

According to the assembly plan of the mixed flow assembly line, it will guide material distribution scheduling. It comprehensively considers the delivery time window requirements of each station to generate the optimal driving route of the delivery vehicle, thereby improving the efficiency and punctuality of logistics distribution and reducing the logistics operation cost.

AGV material distribution route planning is the key to efficient handling. Combined with different constraints, the researchers turned the route planning problem in the manufacturing system into Queuing Theory[1], Traveling Salesman Problem (TSP), and Vehicle Routing Problem (VRP) [2] for research, and the small-scale cases can be solved effectively.

In the material distribution of the manufacturing system, the material distribution of the same station must be completed at one time in the distribution cycle, which is considered as the VRP with time windows planning distribution route in many studies. However, production line-side has a certain inventory as a buffer. In a distribution cycle, the material requirements of the same station can be divided into different time periods for distribution that will be relatively flexible. The demand for the station is delivered by multiple AGVs. And the reasonable disassembly is beneficial to make full use of the loading capacity of the AGV and reduce the number of AGVs and the driving cost. It must be split deliveries when the material demand of the station exceeds the maximum load limit of the AGV[3].

Weikang Fang
The State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: fangweikang@hust.edu.cn

Zailin Guan
The State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan 430074, China
E-mail: zlguan@hust.edu.cn

Each type of parts is different in size, volume, and weight. So the corresponding containers are typically designed for materials in the manufacturing system. It is also delivered in units of containers. We can think that it is possible to split the container in the smallest unit when the station material demand are split. This mode is used in most of the actual logistics distribution of manufacturing systems. But few people study such problems.

Based on the existing research, this paper studies the VRP with the time windows and station requirements material with split deliveries by container. The problem satisfies the conditions of maximum load weight constraint, delivery arrival time windows constraint, station material demand and takes the container as the minimum splitting unit for the distribution route planning of the material in the manufacturing system. In order to solve large-scale problems, we have improved adaptive tabu search algorithm. And this paper uses GA to generate high-quality initial solutions. The multi-domain transformation strategy and dynamic tabu operation are used to enhance the global optimization ability of the algorithm.

2 Problem Description

The VRP with the time windows and station requirements material with split deliveries by container is described as follows. There are K vehicles that can be dispatched. Due to the labor restrictions of the distribution center pickers, tally and loading and unloading workers, each delivery vehicle cannot be operated from the distribution warehouse at the same time. Vehicle blocking during the delivery will be considered. Each station has a time window limit to delivery and the material distribution can be divided into multiple delivery. Distribute or even split in units of containers for each material. Each work station can only be accessed by a single AGV up to once, but can be accessed by multiple AGVs and must meet their needs. Because of the two-way channel, the AGV does not affect each other while driving in each channel. Accidents such as traffic collisions at intersections are not considered.

Under the premise of ensuring normal assembly without interruption, the total time of material distribution is the goal of vehicle route planning. For the material distribution task of a batch of orders in the production workshop, planning the appropriate material distribution route to get the minimum AGVs and departure time. The total time of material distribution is the first objective. Total time includes: 1. The travel time of the AGV from the warehouse to each station; 2. The waiting time for loading and unloading of AGV between station stations; 3. The waiting time for AGV to queue, pick, load and unload in warehouse. Since the expensive price of the AGVs, the secondary objective is to minimize the number of vehicles.

The maximum travel time (distance) of the AGV, the time window of each station, split deliveries of the material requirements of the station, and the maximum load of the vehicle are the comprehensive considerations

Based on the objectives and constraints, a mathematical model based on split deliveries of the material requirements of the station is formulated. An adaptive tabu search algorithm with dynamic tabu list and multi-neighborhood structure is designed to solve the problem.

3 Design of Adaptive Tabu Search Algorithm

Dror and Trudeau proved that the VRP is an NP-Hard problem[4], so the VRP with the time windows and station requirements material with split deliveries by container is also an NP-hard problem, which could be converted to VRP model. Designing heuristic algorithms is a general approach to solving VRP. The tabu search algorithm is the preferred intelligent heuristic algorithm for solving VRP[5].

The tabu search algorithm is improved to solve large-scale problems. The following operations are designed to enhance the global optimization capabilities of the algorithm. The tabu search algorithm has a strong dependence on the initial solution. A good initial solution can make the tabu search search for a good solution in the solution space, while a poor initial solution will reduce the convergence speed of the tabu search. This paper uses GA to generate high-

quality initial solutions, and then uses tabu search to solve the problem for improving the quality and efficiency of the search. Aiming at this problem, we design a variety of domain structures. A random neighborhood transform strategy is used to select a neighborhood for the current solution transformation. Before each operation, a non-zero station is randomly selected in the two lines which be selected randomly for conversion operation (unchanged in other lines). After each neighborhood transformation, the points of the same site in the same line are merged. And the relevant constraints are tested at the same time.The algorithm adopts a strategy that accepts some infeasible solutions, and designs an adaptive penalty mechanism to help jump out of the local optimum to find a better feasible solution.

Based on the characteristics of material demands in assembly workshop, our research could be effectively applied to actual workshop system with reasonable vehicle route plan. The solution has obvious improvements comparing with real data from the real system including shorter travel time and smaller number of AGVs.In the meantime, compared with other algorithms, the improved Tabu Search method could get better solutions so that the problem can be solved effectively.

4 Conclusion

The VRP with the time windows is a type of discrete split deliveries problem, which has wide application value in manufacturing system logistics distribution. Based on the characteristics of material demands in assembly workshop, the VRP with the time windows and station requirements material with split deliveries by container is studied. This thesis constructs the optimization model of vehicle routing problem based on total time of material distribution and the number of vehicles. Tabu search algorithm with adaptive is designed to solve the problem. Multi-neighborhood structure enhances abundance of candidate solutions. The results are compared with other algorithm in the literature, which shows that the proposed algorithm could find solutions with shorter travel time and smaller number of AGVs.

In future research, we can incorporate more uncertain constraints into the basic model such as uncertainty in the variety and quantity of station requirements and random arrival time. The fuzzy processing and simulation optimization methods are used to solve the uncertainty problem and better serve the logistics practice.

Funding

This work is supported by the National Key R&D Program of China：Key Technology Research and Platform Development for Cloud Manufacturing Based on Open Architecture，under Grant No.:2018YFB1702700.

References

1. Hoshino S , Ota J , Shinozaki A , et al. Optimal Design Methodology for an AGV Transportation System by Using the Queuing Network Theory[M]// Distributed Autonomous Robotic Systems 6. Springer Japan, 2007.
2. Fazlollahtabar H , Saidi-Mehrabad M . Optimal path in an intelligent AGV-based manufacturing system[J]. Transportation Letters, 2015, 7(4):219-228.
3. Archetti C , Bianchessi N , Speranza M G . Branch-and-cut algorithms for the split delivery vehicle routing problem[J]. European Journal of Operational Research, 2014, 238(3):685-698.
4. Dror M, Laporte G, Trudeau P. Vehicle routing with split deliveries. Discrete Applied Mathematics. 1994; 50(3): 239–254
5. Cordeau J F , Gendreau M , Laporte G , et al. A guide to vehicle routing heuristics[J]. Journal of the Operational Research Society, 2002, 53(5):512-522.

MISTA 2019

A Deep Reinforcement Learning Based Hyper-heuristic for Container Terminal Truck Routing Problem under Uncertain Service Times

Yuchang Zhang • Ruibin Bai • Rong Qu

1 Introduction

Most of the real-world optimization problems are dynamic. Intelligent optimization algorithms need to be robust and capable of accommodating the uncertainties. In the container terminal truck routing problem, the amount of containers to be handled, service times, or truck travel times may all subject to changes. Intelligent optimization algorithms need to be robust and capable of accommodating these uncertainties. Traditional static and deterministic solution methods based on meta-heuristics, however, show to struggle to solve these dynamic problems.

Deep reinforcement learning (DRL) is one of the effective methods to cope dynamic problems. The deep neural networks in DRL show to be able to perceive and extract features from data automatically, while iteratively increasing its decision-making by 'trial and error' and interactions with the environment. In this research, we investigate deep reinforcement learning incorporated within a selection hyper-heuristic to solve a real-world container terminal truck routing problem with uncertain service times.

2 Literature Review

With the fast developments of technologies on mobile devices and GPS, real-time information, such as customer requests and traffic conditions, needs to be considered continuously over time to make better decisions [2]. The Dynamic Vehicle Routing Problem (DVRP) ([1]), also known as online VRP, can be divided into dynamic problems with deterministic information and dynamic problems with stochastic information. For the former problems, Psaraftis [3] presented a dynamic programming method which, however, due to the curse of dimensionality, can only solve small problem instances. Benyahia and Potvin [4] presented a genetic algorithm which simulated the process of human decision making for solving the VRP with Pick-up and Delivery. This method was then extended to resolve the

Yuchang Zhang
University of Nottingham Ningbo China
E-mail: yuchang.zhang@nottingham.edu.cn
Ruibin Bai
University of Nottingham Ningbo China
E-mail: Ruibin.Bai@nottingham.edu.cn
Rong Qu
University of Nottingham, Nottingham, UK
rong.qu@nottingham.ac.uk

DVRP [5]. For the latter problem, Approximate Dynamic Programming was employed by Powell [6] to deal with the curse of dimensionality problem in dynamic programming. The ADP uses a value function to express the mapping relationship from a state to its value, so that only a set of parameters related to value function need to be saved to obtain the value of each state. Recently, Nazari et al. [7] proposed a reinforcement learning framework, in which they applied a pointer network to cope the uncertain number of customers in the DVRP.

3. Problem Description

This work addresses an increasingly challenging real-world problem faced by Ningbo Port, the busiest port in the world in terms of cargo tonnage. In the problem, a capacity constraint (one vehicle visit at any time) is imposed at each operating point, and the service time at each operating node is uncertain. When determining vehicle dispatching, the service time at each operating node is comparable to the travel time of vehicles, thus cannot be ignored.

3.1 Problem model

In the problem, a fleet of trucks is denoted as $V = \{v_1, v_2, \dots, v_p\}$, with a set of nodes and two types of tasks (loading and unloading). As shown in Figure 1, the trucks are based at the depot D at the beginning of a shift. There are three types of nodes, the depot **D**, the yard cranes $YC = \{yc_1, yc_2, \dots, yc_m\}$, and the ship crane $\mathrm{CR} = \{\mathrm{cr}_1, \mathrm{cr}_2, \dots, \mathrm{cr}_\mathrm{n}\}$. The yard crane and the ship crane can be seen as two different types of customers in the classic VRP. The ship crane is deployed along the coastline and is responsible for hoisting the containers on the trucks to the vessels, or to move the containers on the vessels to the trucks. The yard cranes are deployed next to each small yard area. Similarly, they are responsible for moving containers in the yard area to trucks, or hoisting containers on trucks to the yards. Every moving action is associated with an operating time t. Every pair of yard cranes and ship cranes (yc_i, cr_j), where $yc_i \in YC$, $cr_j \in CR$, is associated with a travel time **t** and a distance **d**.

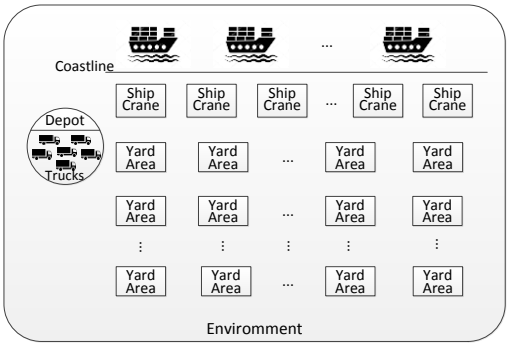


Figure 1: A diagrammatic illustration of the truck routing problem at a container terminal.

3.2 Tasks

Each task consists of three actions:

- Picking up a commodity from the source node.
- Transporting the commodity to the destination node.
- Unloading the commodity at the destination node.

Picking up of containers is associated with a loading time, namely source operation time (SOT); while unloading the container leads to unloading time, namely destination operation time (DOT). The transportation time is the travel time from the source node to the destination

node. A task can thus be represented as a tuple$\langle NO., SN, DN, SOT, DOT \rangle$, where NO. is a sequence number for the task, SN and DN are the source node and destination node, respectively, both of which can be a ship crane or a yard crane which is close to a small yard area in the yard. It should be noted that SOT and DOT cannot be acquired in advance.

There are two types of tasks: loading and unloading. A loading task loads a container from a yard area to a ship, while an unloading task unloads a container from a ship to a yard area.

3.3 Objective and Uncertainty

In this problem, when a ship crane completes a task, and the truck that is dispatched to do the next task in this work queue has not arrived at the ship crane in time, the ship crane will be in a waiting state. The ship crane can start to work again when the designated truck arrives. Since the ship cranes are one of the most critical resources in the port, it is desired to make the most use of them instead of leaving them unoccupied. The objective of this problem is thus to minimise the total waiting time of the ship cranes.

Due to differences in operator proficiency, weather conditions, wind power, and the differences in mechanical equipment itself, there is a considerable amount of uncertainties for the service time, leading to the dynamic vehicle routing problem with uncertain service time.

4. DQN hyper-heuristic framework

In this research, a widely used DRL method, deep Q-network, is investigated. It combines a deep neural network function approximator with the classical Q-learning algorithm to learn state-action value function, and by using a greedy method, a policy can be obtained iteratively [8]. The main body of the framework is the DQN studied in [9], where the actions are replaced with low-level heuristics. Preliminary promising results have been obtained. More details of states, actions, and rewards of the DQN framework, and results will be reported at the conference.

References

1. Pillac, V., et al., *A review of dynamic vehicle routing problems.* European Journal of Operational Research, 2013. **225**(1): p. 1-11.
2. Angelelli, E., et al., *Comparison of policies in dynamic routing problems.* Journal of the Operational Research Society, 2010. **61**(4): p. 686-695.
3. Psaraftis, H.N., *A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem.* Transportation Science, 1980. **14**(2): p. 130-154.
4. Benyahia, I. and J.-Y. Potvin, *Decision support for vehicle dispatching using genetic programming.* IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 1998. **28**(3): p. 306-314.
5. van Hemert, J.I. and J.A. La Poutre, *Dynamic routing problems with fruitful regions: Models and evolutionary computation.* Parallel Problem Solving from Nature - Ppsn Viii, 2004. **3242**: p. 692-701.
6. Powell, W.B., *What you should know about approximate dynamic programming.* Naval Research Logistics (NRL), 2009. **56**(3): p. 239-249.
7. Nazari, M., et al., *Deep Reinforcement Learning for Solving the Vehicle Routing Problem.* arXiv preprint arXiv:1802.04240, 2018.
8. Mnih, V., et al., *Playing atari with deep reinforcement learning.* arXiv preprint arXiv:1312.5602, 2013.

9. Mnih, V., et al., *Human-level control through deep reinforcement learning.* Nature, 2015. **518**(7540): p. 529.

MISTA 2019

A branch-and-price algorithm for bunkering tanker

Feng Li · Dong Yang · Shuaian Wang

1 Problem Description

Almost of the extant academic literature on refueling discuss refueling port choice. However, in this talk, we consider a routing and scheduling problem of bunkering tankers for refueling ships.

At the beginning of a planning horizon, there exists a single port, denoted by 0, a set of anchorages, denoted by $\mathcal{A} = \{1, 2, \ldots, |\mathcal{A}|\}$, and a fleet of homogeneous bunkering tankers, denoted by $\mathcal{V} = \{1, 2, \ldots, |\mathcal{V}|\}$. In the planning horizon, let n_i denote the number of vessels and $N_i(|N_i| = n_i)$ the set of the vessels berthing in anchorage $i \in \mathcal{A}$ to refuel by the bunkering tankers, where $n_1 + n_2 + \ldots + n_{|\mathcal{A}|} = n$. Vessel j in N_i is denoted by (i, j), has a required quantity of q_{ij} for fuel, has a required refueling time of s_{ij}, has a profit of p_{ij}, and is associated with a time window $[a_{ij}, b_{ij}]$. This implies that the refueling for vessel (i, j) must start in the time window $[a_{ij}, b_{ij}]$, where a_{ij} and b_{ij} are the earliest time and latest time to refuel, respectively. As is often the case in practice, we assume that the port has a time window $[a_0, b_0]$, where a_0 is the earliest time to refuel at the port for bunkering tankers and b_0 is the latest time that all bunkering tankers return to the port. Without loss of generality, we let $a_0 = 0$. We also make the simplifying assumption that at the beginning of the planning horizon each bunkering tanker with a capacity of Q units is empty. Each bunkering tanker are first refueled fuel at the port and then visits vessels for refueling. Each vessel can be served at most once. A bunkering tanker must wait if it arrives at vessel (i, j) before a_{ij} for refueling. Each bunkering tanker has a refueling time of s_0 at the port, where at most one bunkering tanker is refueled at a time because of the capacity of the port and the shortage of refueling devices at the port. The travel time of a bunkering tanker

Feng Li
Huazhong University of Science and Technology, Wuhan, China,
E-mail: li_feng@hust.edu.cn

Dong Yang
The Hong Kong Polytechnic University, Hung Hom, Hong Kong,
E-mail: dong.yang@polyu.edu.hk

Shuaian Wang
The Hong Kong Polytechnic University, Hung Hom, Hong Kong,
E-mail: wangshuaian@gmail.com

from the location of anchorage i to the location of anchorage k is denoted by t_{ik}, for $i, k \in \mathcal{A}$. The travel time of a bunkering tanker from the port to the location of anchorage $i \in \mathcal{A}$ is denoted by t_{0i}. In this paper, we assume that the travel time of one bunkering tanker from the location of vessel $(i, j_1) \in N_i$ to the location of vessel $(i, j_2) \in N_i$ is 0 for $i \in \mathcal{A}$. The problem considered in this paper is to serve a subset of vessels in the planning horizon so that a certain objective is optimized.

2 Results

We first formulate the problem as a mixed integer linear programming model (MILPM) and prove that the problem is strongly NP-hard. Then we develop a branch-and-price algorithm to solve the problem to optimality. Furthermore, we design an optimal algorithm for a special case of the problem. Finally, computational experiments demonstrate that our branch-and-price algorithm is better than solving the MILPM in terms of computational time and the number of instances solved to optimality in a time limit.

3 Related Literature

This talk belongs to a small but growing literature investigating ship refueling problems. However, ship refueling problems are mainly considered from ship operators perspective, that is, when and where to refuel. For example, Plum et al. (2014) proposed a mixed integer programming model for bunker purchasing with contract and designed a column generation algorithm to solve the model. Wang and Meng (2015) considered the ship speed and refueling decision in a liner shipping network and developed a mixed integer programming model. Meng et al. (2015) presented a joint tramp ship routing and bunkering (JSRB) problem which determines how to route ships to carry the cargoes and the amount of fuel to purchase at each port, in order to maximize the total profit. A tailored branch-and-price (B&P) solution approach is proposed. Zhen et al. (2017) designed a dynamic programming algorithm to decide at which ports and how much fuel the liner ship should be refueled under stochastic fuel consumption in each leg and stochastic fuel price at each port. The problem we will discuss in this talk is different from the previous studies. In general, the decision maker is not a shipping carrier but a fuel supplier. If there are too many vessels waiting to be refueled, the fuel supplier will be incapable of servicing all vessels within their time windows; in other words, some vessels are rejected by the fuel supplier. We maximize the profit of a fuel supplier by determining the choice of vessels to refuel and the routes and schedules of bunkering tankers.

References

Meng Q., Wang S., Lee C.Y. A tailored branch-and-price approach for a joint tramp ship routing and refueling problem, Transportation Research Part B 72: 1-19 (2015).

Plum C.E.M., Jensen P.N., Pisinger D. Bunker purchasing with contracts, Maritime Economics & Logistics, 16: 418-435 (2014).

Wang S., Meng Q. Robust bunker management for liner shipping networks, European Journal of Operational Research, 243: 789-797 (2015).

Zhen L., Wang S., Zhuge D. Dynamic programming for optimal ship refueling decision, Transportation Research Part E 100: 63-74 (2017).

MISTA 2019

An integrated machine-learning and optimization approach for a vehicle dispatching and routing problem in logistics industry

Shiqiang Liu • Weixu Pan

Abstract Vehicle dispatching and routing management is a vital issue to boost and sustain the rapid growth of logistics companies. In this paper, a real-world vehicle dispatching and routing problem with uncertainties is studied to estimate and optimize the total fuel consumption of vehicles. A big data analytics framework is designed to alleviate the imbalance between supply and demand for the rational use of fuel. Furthermore, we develop an integrated Machine Learning (e.g., principal components analysis and support vector machine) and Optimization (e.g., mathematical formulation and heuristics) approach. Computational results validate that the proposed methodology is promising to improve the efficiency of express delivery process.

Shiqiang Liu, Weixu Pan
Fuzhou Unviersity
E-mail: samsqliu@fzu.edu.cn | N190710007@fzu.edu.cn

MISTA 2019

Modifying Kruskal algorithm to solve open loop TSP

Pasi Fränti • Henrik Nenonen

Abstract Traveling salesman problem (TSP) has been widely studied for the classical closed loop variant. However, only little attention has been paid to the open loop variant. Open loop solution represents also a spanning tree, but not necessarily the minimum spanning tree (MST). However, solution for MST contains lots of information useful for solving TSP as well. In this paper, we modify Kruskal algorithm to find heuristic solution for TSP instead. The basic variant achieves 4.6% gap, and its randomized variant 0.65% gap with our O-Mopsi game datasets, on average. The algorithm is studied mainly to fulfill our curiosity, but the results show the method has some usefulness for providing a quick approximation for TSP.

1 Introduction

In the classical *closed loop* variant of *travelling salesman problem* (TSP), the goal is to visit all the nodes and then return back to the start node so that the length of the tour is minimized. A slightly different variant, referred as *open loop TSP*, we can skip the return to the start node. Both variants are known to be NP-hard [1]. Finding the optimum solution fast is therefore not realistic except for very small size inputs.

The open loop variant appears in *O-Mopsi* (http://cs.uef.fi/o-mopsi/) mobile-based orienteering game where the player needs to find a set of real-world *targets* with the help of GPS navigation [2]. Game playing includes three challenges: (1) navigating to the next target, (2) deciding the order of visiting the targets, (3) physical movement. Unlike in the classical orienteering, in which the order of visiting the targets is fixed, O-Mopsi players must optimize the order while playing the game. This corresponds to a special case of the *orienteering problem* [3]. Finding the optimum order is not required but desirable in order to minimize the tour length [4]. According to [2], among players who completed the tour, only 14% managed to find the best visit order, i.e. solving the TSP problem involved.

Pasi Fränti
School of Computing, University of Eastern Finland
E-mail: pasi.franti@uef.fi

Henrik Nenonen
School of Computing, University of Eastern Finland
E-mail: henrikne@student.uef.fi

There is an algorithmic connection between MST and TSP worth to explore. The solution of the open loop TSP is also a spanning tree, but not necessarily the minimum one, see Fig. 1. Thus, both solutions have equal number of links. This connection is interesting because, while TSP is NP-hard, MST can be solved in polynomial time using greedy algorithms like Prim and Kruskal. This connection has been utilized at least in three different ways.

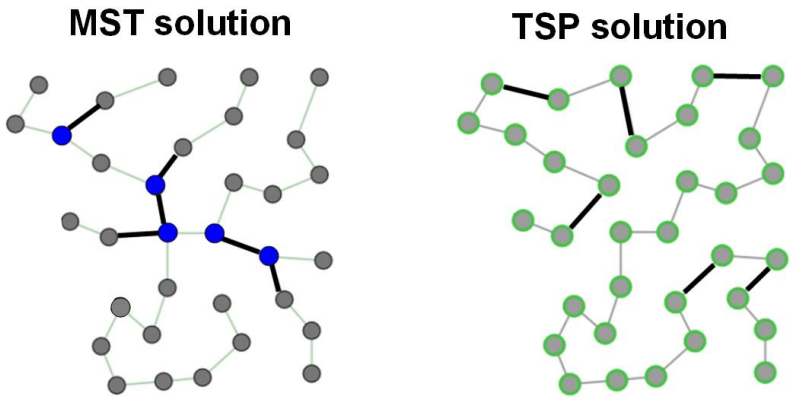


Figure 1: MST and open loop TSP solutions for the same problem instance.

First, classical *Christofides algorithm* [5, 6] starts by creating an MST, and then adds more links so that Euler tour becomes possible. Final solution is obtained by removing nodes from the tour that are visited more than once. Second, MST has also been used to estimate lower limit for the optimal TSP solution [7]. Third, the difficulty of the TSP instance was estimated in [8] by counting the number of branch nodes that branches the tree. The more branches, the more difficult is the instance to be solved both by human and by computer.

In this paper, we show another connection between MST and TSP. In specific, we modify classical *Kruskal algorithm* to create a heuristic solution for the open loop travelling salesman problem. Kruskal algorithm operates on a spanning forest by greedily merging two spanning trees increase the total cost of the spanning forest least. We introduce a simple constraint that allows merging two sub-solutions using only their end points. This not only prevents branches to appear in the tree but also reduces the number of possible links to be considered for the merge. Thus, it enforces the final result to be a solution for the open loop travelling salesman problem and allows potentially more efficient implementation.

Our experiments will show that the proposed *Kruskal-TSP algorithm* provides average gap of about 4.8 % from the optimum. Although this is still rather far from the optimum, it can be useful for providing reasonable estimate for the TSP solution. The algorithm is also fast and provides comparative solutions to other heuristics. It also has no restrictions like Christofides algorithm, which is limited to work only on metric data [5]. In addition, we also propose repeated randomized variant of the algorithm, which further reduces the gap down to 0.77%, on average.

2 Kruskal-TSP algorithm

2.1 Basic variant

The proposed Kruskal-TSP algorithm works similarly as classical Kruskal by always taking the smallest link that does not create a loop. While Kruskal produces a spanning forest (a set of spanning trees), our Kruskal-TSP produces a set of chains (sub-tours). The only limitation we need to add is to allow the merges to be made using only the end points of the two chains. This

reduces the number of choices for the merge, and also causes sub-optimal spanning tree to be created; but being chain it gives a solution to the TSP as well.

Pseudo code of the modified Kruskal algorithm is shown in Figure 2. In the beginning, every node is assigned to its own set. The links are sorted and then processed from shortest to longest. Every link that connects two distinct sets *from their end points* are merged; thus, avoiding cycles and *branches*. Merges are continued until only one set remains. Efficient implementation can reach $O(N^2)$ time complexity where N is the number of nodes.

The effect of the modification is shown in Figure 3. The positive side is that the number of choices for the next merge is significantly reduced but the negative effect is that the optimality cannot be anymore guaranteed.

```
Kruskal-TSP(V, E): RETURN T
FOR ALL v∈V
    CreateSet(v);
P←∅;  SORT(E);   // according to increasing weights
FOR each (u,v)∈E DO
    IF Set(u)≠Set(v) AND End(u) AND End(v) THEN
      UNION(u,v);
      P←P∪{(u,v)};
Return P;
```

Figure 2: Pseudo code of the Kruskal-TSP algorithm.
Modifications from standard Kruskal are marked in blue.

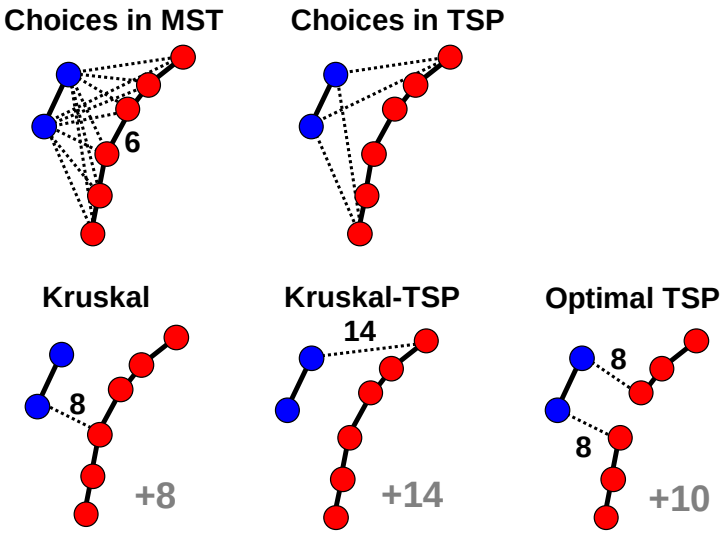


Figure 3: Difference of the merge in Kruskal and in the Kruskal-TSP.

Various similar heuristics were also summarized in [9] but without any details or experimental studies. Among them, Rosenkrantz et al. [10] considered two simple heuristics that are closest to our Kruskal-TSP. The first one is called *Myopic*. It starts from an arbitrary node and then subsequently adds always the nearest node to the one that was last added. The second one, called *Insertion algorithm*, adds the one that increases the tour least. In other

words, addition is allowed at both ends of the tour. This idea can be considered as modification of Prim's MST algorithm, and therefore, is closest to our proposal.

The Kruskal-TSP algorithm is pretty much like single link agglomerative clustering except the linkage criterion must use only the end points. The benefit is that the merge is simpler to implement than in case of single link. The drawback is that it does not guarantee optimality. In the same way as with agglomerative clustering, early merges cannot be reversed and can cause sub-optimal choices later.

An example of the algorithm is given in Fig. 4 where selected steps of the algorithm are shown. The 6th merge is locally optimum choice at the time, but this link is not included in the optimal solution. It will cause sub-optimal choice for the 15th merge. In general, most errors happen at the later stages of the algorithm when there are less candidate pairs to be merged.

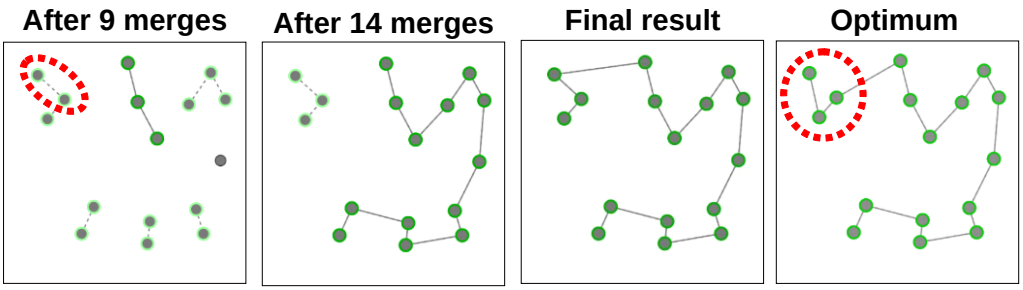


Figure 4: Example of the Kruskal-TSP algorithm with minor difference on the top-left corner.

2.2 Randomized variant

Since the algorithm is relatively fast, it is possible to improve its performance by repeating it several times. This improvement is inspired by [11] where it was shown that k-means clustering algorithm can be significantly improved simply by repeating the algorithm 100 times. The only condition is that the (initialization) algorithm contains randomness so that it produces (slightly) different results after every repeat. The final solution is the best result out of the 100 trial runs.

In our case, we add randomness as follows. Instead of always selecting the shortest link for the merge, we find 3 shortest links and select one of them by random choice. The consequence is that there will be making several sub-optimal choices. However, there are reasons why this can be done. First, the final result will be an approximation anyway, so it is not necessary that every link is optimally chosen. Second, our input is a complete graph, and therefore, there are often several almost equally good links to choose from.

The amount of variation in the process can be increased by changing the parameter 3. The more choices, the more variations there will be among the solutions. However, the average quality of the solutions tends to decrease with the randomness. It is therefore likely that a small value would work better. We tested with the values of 2-5 but the results were virtually the same in all cases. The optimal choice would depend on the size of the problem instance (N), and on the number of repeats.

2.3 Relationship between the open and closed loop variants

The open and closed loop cases are closely related. However, it is not trivial how to convert the solution from one to the other. For example, removing the longest link from the closed loop solution does not necessary lead to the optimum solution for the open loop case; and vice versa; see Fig. 5.

However, it is possible to modify the open loop problem instance to create a closed loop problem instance. One can then solve the problem using algorithm for the closed loop solution

and derive solution for the open loop case [1, 12]. This also indicates that the open loop case is a special case of the closed loop.

The modification of the open loop solution happens as follows. First, we add artificial pseudo node with very large distance to all other nodes. The consequence of this is that two links with very large cost must be used, and that one normal link dropped. The nodes connecting to the pseudo node will act as the terminal nodes of the open loop solution. The final solution is then obtained by removing this phantom node and the links connecting to it. The reduction to the other way (closed loop to open loop) is less obvious. However, the proposed Kruskal-TSP can be applied to the closed loop version as such, if so wanted.

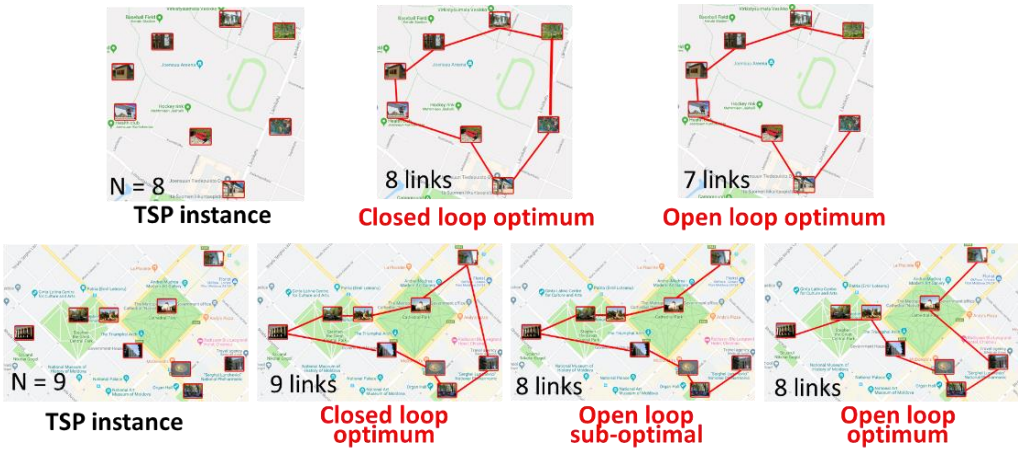


Figure 5: Difference between open and closed loop TSP. [13]

3 Experimental results

We use two open loop datasets found here: http://cs.uef.fi/o-mopsi/datasets/. The O-Mopsi dataset contains TSP instances used in the O-Mopsi game [2]. There are 145 game instances everywhere in the world; although most of them are in Joensuu, Finland. The Dots dataset contain 4226 computer-generated TSP instances that were used for experimenting human problem solving skills in [12]. Optimal ground truth solutions are given for all instances. To evaluate the performance of the algorithms, we calculate the *gap*, i.e. the relative difference of the solution provided by the algorithm, and the optimal solution:

$$gap = \frac{\left| L_{Kruskal} - L_{opt} \right|}{L_{opt}} \quad (1)$$

Table 1: Datasets used in the experiments.

Dataset:	Reference:	Distance:	Instances:	Problem sizes:
O-Mopsi	[2]	Haversine	145	4-27
Dots	[13]	Euclidean	4226	4-31

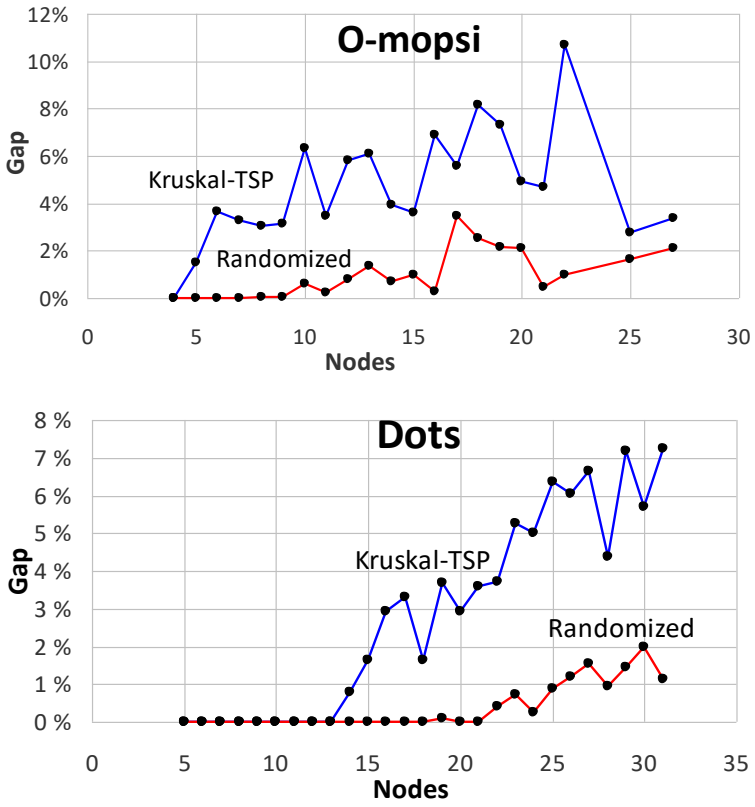


Figure 6: Results for the O-Mopsi (above) and Dots (below) as a function of problem size.

Results are plotted in Figure 6 for the O-Mopsi datasets with respect to the number of nodes. We can see that the Kruskal-TSP reaches the optimal solution only with the smallest dataset with 4 nodes. However, the gap remains less than 10% in almost all cases, being about 5%, on average (O-Mopsi) and <3% (Dots). The randomized variant reduces the gap further down to 0.77% (O-Mopsi) and 0.26% (Dots).

Comparison to Christofides [6] and the local search algorithm in [13] are summarized in Table 2. We apply Christofides using the added pseudo node strategy as explained in Section 2.3. The results show that Kruskal-TSP is better than Christofides but inferior to the local search. Optimal solution is found 35% (basic variant) and 74% (randomized) with the O-Mopsi data while local search finds it 100% times and start to struggle only with larger number of nodes. The results show the practical limitations of the proposed Kruskal-TSP. Nevertheless, we have shown by proof-of-concept that Kruskal algorithm can be modified to find reasonable approximations for TSP as well with real-life data.

Table 2: Summary of the results (gap).

Algorithm:	**Reference:**	**O-Mopsi:**	**Dots:**
Christofides	[6]	12.77%	9.10%
TSP-Kruskal	(proposed)	4.63%	2.66%
Randomized TSP-Kruskal	(proposed)	0.65%	0.26%
Local search	[13]	0.00%	0.001%

4 Conclusions

We introduced algorithm for solving open loop variant of TSP by modifying classical Kruskal algorithm. Although many heuristics exist in literature, extending Kruskal has a special interest as it shows the relationship between an NP-hard problem (TSP), and easier problem (MST).

The proposed Kruskal-TSP can be used when one needs an approximation of the TSP fast. The results also show that the randomized variant reaches <1% gap to the optimum, on average with our datasets. However, the performance weakens with the number of nodes in the graph. This is likely to leave the algorithm merely as theoretical interest to show how good results can be reached by such simple heuristics. Nevertheless, having reasonable approximation in real-time can still be useful. For instance, a good upper bound can also be critical when finding exact solutions for TSP and vehicle routing problems using branch-and-cut or by enumeration [14, 15]. The randomized Kruskal-TSP could also be used to provide input generation for a genetic algorithm instead of using merely random solution.

In principle, the algorithm can also be applied to closed-loop TSP but not recommended. The reason is that there are no alternatives for the last merge, as we just need to connect the two end points whatever they are. Effectively, it is almost like linking two random nodes, which can result in adding one arbitrary long link.

References

1. C.H. Papadimitriou, C.H. The Euclidean travelling salesman problem is NP-complete. Theoretical Computer Science, 4 (3), 237-244 (1977)
2. P. Fränti, R. Mariescu-Istodor and L. Sengupta, O-Mopsi: mobile orienteering game for sightseeing, exercising and education, ACM Trans. on Multimedia, Computing, Communications, and Applications, 13 (4), 56:1-25, (2017)
3. P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden: The orienteering problem: A survey. European Journal of Operational Research. 209, 1, 1–10 (2011)
4. H.H. Chieng, N. Wahid, A performance comparison of genetic algorithm's mutation operators in n-cities open loop travelling salesman problem. Recent Advances on Soft Computing and Data Mining, Springer (2014)
5. M.T. Goodrich, R. Tamassia, 18.1.2 The Christofides approximation algorithm, Algorithm Design and Applications, Wiley, pp. 513-514. (2015)
6. N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU, 1976.
7. C.L. Valenzuela, A.J. Jones, Estimating the Held-Karp lower bound for the geometric TSP, European Journal of Operational Research, 102, 157-175 (1997)
8. L. Sengupta and P. Fränti, Predicting difficulty of TSP instances using MST, IEEE Int. Conf. on Industrial Informatics (INDIN), 847-852, Helsinki (2019)
9. G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59.2, 231-247. (1992)
10. J. Rosenkrantz, R.E. Stearns, P.M. Lewis II, An analysis of several heuristics for the travelling salesman problem, SIAM journal on Computing, 6, 563-581 (1997)
11. P. Fränti and S. Sieranoja, How much k-means can be improved by using better initialization and repeats?, Pattern Recognition, 93, 95-112 (2019)
12. L. Sengupta, R. Mariescu-Istodor and P. Fränti, Planning your route: where to start?, Computational Brain & Behavior, 1 (3-4), 252-265, (2018)
13. L. Sengupta, R. Mariescu-Istodor and P. Fränti, Which local search operator works best for open loop Euclidean TSP, *Applied Intelligence*, 9 (19), 3985, (2019)
14. L. Costa, C. Contardo, G. Desaulniers, Exact branch-price-and-cut algorithms for vehicle routing, Transporation Science, 53, (2019)
15. G. Homsi, R. Martinelli, T. Vidal, K. Fagerholt, Industrial and tramp ship routing problems: closing the gap for Real-scale instances, Arxiv. 1809.10584, (2018)

MISTA 2019

Classification-based algorithm selection for a multiprocessor scheduling problem

Roland Braune

1 Problem statement

The subject of this contribution is a particular kind of multiprocessor scheduling problem. A set of n tasks $T = \{1, \ldots, n\}$ is to be processed on a set of m identical parallel processors. Each task $i \in T$ consists of n_i subtasks or task "slices". Such a sub-task is identified by a pair (i, j), with $i \in T$ and $j \in \{1, \ldots, n_i\}$. The number of processors to be allocated simultaneously for the execution of a task slice is denoted by $size_{i,j}$, also commonly known as the *width* of a task (slice) [4]. All slices (i, j) that make up a task $i \in T$ have unit processing time, that is, $p_{ij} = 1$, $i \in T$, $j \in \{1, \ldots, n_i\}$, and share the same width, hence, $size_{i,1} = size_{i,2} = \ldots = size_{i,n_i}$. Consequently, it is possible to specify the width on the level of outer tasks $i \in T$, yielding $size_i = size_{i,1} = \ldots = size_{i,n_i}$. The slices of a task are organized into chain-like structures by defining appropriate, generalized precedence constraints. Those constraints are enforced by imposing minimum time lags (delays) on every two sub-tasks of one and the same outer task. Since preemption of tasks is not allowed, all slices of a task are also subject to maximum time lags. Given a task $i \in T$, let $l_{i,j,j'}$ denote the end-to-start time lag (delay) between two task slices (i, j) and (i, j'), $(j, j') \in \{1, \ldots, n_i\} \times \{1, \ldots, n_i\}, j \neq j'$. Assuming that $s_{i,j}$ and $C_{i,j}$ denote the start and completion time of a task slice (i, j), respectively, one can define the constraints $s_{i,j'} \geq C_{i,j} + l_{i,j,j'}$, for a task $i \in T$ and any pair of slice indices j and j' as defined above.

For the problem at hand, $C_{i,j} = s_{i,j}+1$ for all (unit-time) task slices (i, j), $l_{i,j,j+1} = -1$, for $j \in \{1, \ldots, n_i-1\}$ and $l_{i,j,j-1} = -2$, for $j \in \{2, \ldots, n_i\}$. Note that the resulting precedence constraints most closely resemble classical chain precedence constraints, with the exception that two subsequent task slices may start at the same time and are not allowed to be scheduled more than one time unit apart. Chain precedence constraints have already been discussed in the multiprocessor scheduling literature [2], but not in conjunction with maximum time lags.

Roland Braune
Department of Business Decisions and Analytics
University of Vienna
Oskar-Morgenstern-Platz 1, 1090, Vienna, Austria
E-mail: roland.braune@univie.ac.at

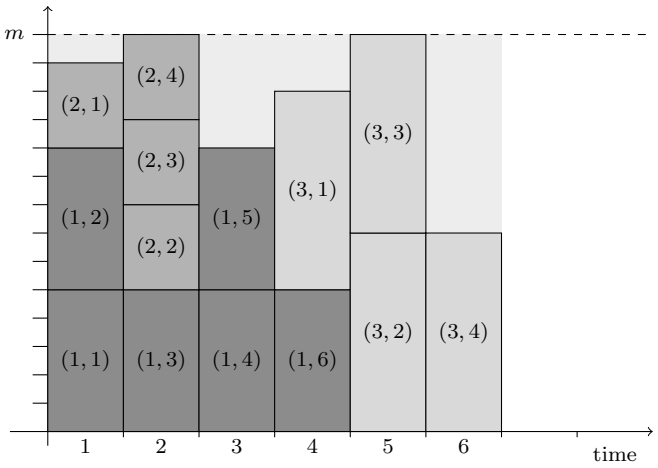


Fig. 1 Scheduling multiprocessor task chains subject to minimum and maximum time lags.

Figure 1 shows a multiprocessor schedule for $m = 14$ processors, based on three different task chains, with 6, 4 and 4 slices (sub-tasks), respectively. The widths are given as $size_1 = 5$, $size_2 = 3$, and $size_3 = 7$. It can be seen that the processor allocation changes three times for chain 1, once for chain 2 and twice for chain 3. The number of sub-tasks of a particular chain that are processed simultaneously is not limited, but at least one sub-task has to be processed at any time during the processing time window of a chain because of the implicit non-preemption constraint.

The objective is to minimize the total weighted completion time (TWCT), given by $\sum_{i,j} w_{ij}C_{ij}$, where w_{ij} denotes the weight at the sub-task level, with $w_{ij} = size_{ij}$, for all $i \in T$ and $j \in \{1, \ldots, n_i\}$.

Relying on the common three-field notation of [6], the problem under consideration is of type $P \mid chains(l); size_i; p_i = 1; w_i = size_i \mid \sum w_i C_i$.

As far as the problem's complexity is concerned, it can be considered a generalization of $P \mid size_i; p_i = 1 \mid \sum C_i$, that is, the multiprocessor scheduling problem with unit time tasks and total completion time objective. The latter problem has been proven to be NP-hard in the strong sense [5], therefore also implying strong NP-hardness for the problem at hand.

2 Solving instances with "small-sized" items

For a relaxation of the above defined problem that omits the precedence constraints, lower bounds and a branch-and-bound algorithm have recently been proposed, based on a bin packing reformulation [3]. Currently, an extension of this exact approach to the full, precedence-constrained version of the problem is under development. During experimentation, it turned out that problem instances involving very small-sized tasks, that is, instances with $size_{ij} \in [1, 0.5m]$ are very hard to solve for the branch-and-bound algorithm. This is not a surprise, since the employed lower bounding schemes are still built upon a bin packing formulation of the problem and thus the bound quality depends on the bulkiness of the tasks (= items). The higher the percentage of tasks with $size_{ij} > 0.5m$, or even $size_{ij} > (2/3)m$, the tighter the bound, and vice versa.

On the other hand, it also became apparent that optimal solutions for small-sized instances frequently coincide with the *fractional* lower bound of the associated bin packing problem. Letting w_{ij} denote the size of an item (corresponding to a task slice), the fractional lower bound is obtained from completely filling the first $\bar{k} = \lfloor \sum_{i,j} w_{ij}/m \rfloor$ bins, if necessary by splitting items. The remaining item load, i.e., $\sum_{i,j} w_{ij} - \bar{k} \cdot m$, is placed into an additional bin. Note that each bin corresponds to a time period in the original multiprocessor scheduling problem. It is easy to see that this kind of (fractional) packing/scheduling leads to the smallest possible total weighted completion time. If one can find a *feasible*, *non-fractional* schedule that exhibits this property, then it is clearly an optimal solution to the underlying problem.

Extensive computational experiments have shown that the number of instances actually permitting the construction of such a non-fractional, minimum TWCT schedule increases markedly with $size_{ij} \leq 0.3m$. Therefore, instances of this type shall be the focus of the remaining part of this paper.

Assume that one knows with *certainty* that such a feasible schedule exists, and let $\mathcal{T}$ denote the set of all task slices. Then an obvious approach would consist of trying to create such a schedule in a chronological fashion, starting at time zero. In each time period $0 \leq k \leq \bar{k}$, the goal is to find a subset $\mathcal{T}^k \subset \mathcal{T}$ such that $\sum_{(i,j) \in \mathcal{T}^k} w_{ij} = m$. Essentially, this procedure solves a series of *subset sum* problems [7]. Of course, the composition of the subset sum problems at later stages of that procedure depend on which tasks have been selected at earlier stages. In other words, it actually matters which of the potentially numerous optimal solutions to a single-stage problem is placed into the schedule. For this reason, the key to the effectiveness of the proposed procedure is (limited) backtracking based on enumerating subset sum solutions. Despite its relative simplicity, this approach proved strikingly powerful for instances with $size_{ij} \leq 0.3m$. Since the branch-and-bound algorithm is based on feasible sets [3], it suffers from an increased set cardinality and thus exploding combinatorics in the small-size scenarios, leading to considerably longer computation times in many such cases. Note that for $0.3m \leq size_{ij} \leq 0.5m$, and particularly $size_{ij} > 0.5m$, the occurrence of minimum TWCT solutions becomes less likely and the branch-and-bound algorithm gains in

3 Predicting the existence of non-fractional, minimum TWCT schedules

Since there is no "oracle" available that may tell with certainty whether a non-fractional, minimum TWCT schedule exists for the considered multiprocessor scheduling problem with precedence constraints, a predictive, learning-based approach is the most obvious way of deciding up front whether or not to apply the iterative subset sum procedure described in Section 2. The design of that procedure is not versatile in the sense that it can only find solutions that lead to 100% utilization in all but one time periods. Consequently, applying the procedure in cases where no such feasible solution exists should be avoided.

To address this prediction-oriented task, a classification problem was defined in the following way: Features (predictors) were extracted from each problem instance, relying on a broad spectrum of different aspects, including

- Relative task size (relative to m): min, max, average, std. dev., quantiles
- # of distinct task sizes ($\approx$ number of chains)
- Chain length: coefficient of variation

- Subset sums: density, relative average, coefficient of variation
- # of feasible sets with maximum utilization
- Is there a subset sum that amounts to the residual load in the last time period of a fractional, minimum TWCT schedule? (binary feature)
- The number of time periods with 100% utilization, given disjoint feasible sets
- Task size distribution (histogram with relative frequencies)

Note that the subset sum-related features are built upon the notions of "concentration" and "density" [1]. The labels for the training data are represented by simple binary digits, indicating whether or not a non-fractional, minimum TWCT schedule exists. The labels are computed using the dedicated branch-and-bound algorithm.

4 Preliminary computational results

Preliminary computational experience was obtained using a set of instances with 10 chains, between 3 and 8 items per chain, yielding a total number of 50 task slices with $size_{ij} \in [1, 0.3m]$. The number of processors m was set to 500. The label distribution for that instance set is almost perfectly balanced, hence there is no bias with regard to either "yes" or "no" instances. A total number of 10000 examples were generated, 60% of which were used as a training set and the remaining 40% as a test set. The common Python analytics framework `scikit-learn` (version 0.20.2) was employed for data processing and fitting various different classifiers, including support vector classification (linear/non-linear), logistic regression, decision trees, and a neural network.

Even without further feature processing, engineering, filtering, or dimensionality reduction, a prediction accuracy of around 90% could be obtained, across the whole range of applied classifiers, linear and non-linear ones. These results provide first evidence for the potential of the proposed prediction approach. The successful application to other instance sets involving more tasks and/or longer chains underpin the extrapolation/generalization capabilities of the trained classifiers.

References

1. Austrin, P., Kaski, P., Koivisto, M., Nederlof, J.: Subset Sum in the Absence of Concentration. In: E.W. Mayr, N. Ollinger (eds.) 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 30, pp. 48–61. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015). DOI 10.4230/LIPIcs.STACS.2015.48
2. Blazewicz, J., Liu, Z.: Scheduling multiprocessor tasks with chain constraints. European Journal of Operational Research **94**, 231–241 (1996)
3. Braune, R.: Lower bounds for a bin packing problem with linear usage cost. European Journal of Operational Research **274**(1), 49–64 (2019). DOI https://doi.org/10.1016/j.ejor.2018.10.004
4. Drozdowski, M.: Scheduling for Parallel Processing. Computer Communications and Networks. Springer Verlag London (2009)
5. Drozdowski, M., Dell'Olmo, P.: Scheduling multiprocessor tasks for mean flow time criterion. Computers & Operations Research **27**(6), 571–585 (2000)
6. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Operations Research **5**, 187–326 (1979)
7. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons, New York (1990)

MISTA 2019

An Adaptive Large Neighbourhood Search Algorithm for Shift Assignment in Shift-Based Businesses

Simon Kløve Christensen

1 Introduction

Scheduling of personnel is still a manual and time consuming process in many businesses. This is especially true for businesses such as restaurants and bars, where the responsibility of scheduling is one among many put upon the manager.

The general problem of assigning a set of well defined shifts within a planning period to a workforce of hourly waged employees is considered here. This type of problem typically arises in shift-based businesses where the set of shifts to be assigned often only changes slightly from planning period to planning period, but the availability of part-time workers may change significantly. In this context a shift-based business is defined as a business that uses hourly waged staff to cover a significant part of their demand. Further, a well defined shift has a fixed start and end time, and a set of required competences. Each employee has a set of competences, a weekly target number of working hours, a set of time intervals of preferred working hours, and a set of time intervals where they are unavailable to work. This gives rise to the multi-objective optimization problem of trying to maximize shift coverage and fulfilled working time preferences of employees, while respecting relevant working time rules.

2 Mathematical Model

The full mathematical model is not included in this abstract, instead a brief description is given. A multi-objective mixed integer programming model is formulated with the solution value defined as the weighted sum of the following objectives:

- Minimize the number of uncovered shifts.
- Maximize the total number of shift hours assigned to employees in available periods.
- Minimize the deviation from target working hours for employees.

The model implements the following basic constraint types:

- An employee must have the required competences to work a shift.
- An employee cannot work two overlapping shifts.
- A constraint for the max allowable negative deviation from the weekly target hours for each employee.
- A constraint for the maximum allowable positive deviation from target hours.

Simon Kløve Christensen
Technical University of Denmark
E-mail: siklc@dtu.dk

Further, businesses may have to respect a combination of the following working time rules:

- Maximum allowable working time per day.
- Minimum time off work between shifts.
- Maximum number of working days in one calendar week.
- Maximum allowable number of work hours in a period of n days.
- Maximum number of consecutive working days.
- Minimum of x weekends off in a period of y weeks.

3 Adaptive Large Neighbourhood Search

For large test instances, the above MIP model has proven to be difficult to solve using IBM's CPLEX solver, by running out of memory before obtaining a feasible solution for one of the test instances. Therefore a metaheuristic solution approach has been implemented. The Adaptive Large Neighbourhood Search (ALNS) Algorithm [1] was chosen as it has proved efficient on the related problem of timetabling [2], but has seen limited application to personnel scheduling problems. The implementation of ALNS follows the framework established by Ropke and Pisinger [1] and implements the following destroy and repair methods:

- *Destroy* n random shifts.
- *Destroy* all shifts on day t.
- *Destroy* all shifts assigned to employees $i \in I$.
- *Destroy* the n worst shift assignments.
- *Global greedy repair*: while a feasible assignment yielding an improvement to the objective score exists, make the most improving shift assignment.
- *Local greedy repair*: randomly go through each unassigned shift and make the most improving feasible assignment, if any exist.
- *Repair greedily by number of regrets*: Make the shift assignment that has the largest difference between best possible choice of employee and second best choice. Similar versions are implemented for third and fourth best choices.
- *Repair randomly*. Randomly makes a feasible assignment, if any exists for each unassigned shift. Only used for creating initial feasible solution.

The ALNS is implemented with an acceptance criteria borrowed from simulated annealing. This makes the algorithm more likely to initially accept an iteration that worsen the solution value, but as the algorithm progresses, this become less and less likely.

4 Results

The performance of the ALNS algorithm was evaluated on 7 data sets obtained from real companies for a scheduling period of 2 months. All tests where carried out on a laptop with a 2,3 Ghz Intel Core i7 processsor with 4 cores and 16 GB of RAM. An overview of the test instances is shown in table 1. The ALNS algorithm was given a fixed running time of 900 seconds and the average solution value over 10 runs reported. Table 2 shows that the ALNS is able to find solutions with a gap of less than 10 % for most of the test instances. The ALNS was further able to find feasible solutions to the test instance D for which CPLEX was unable to solve and even find an optimal solution before running out of memory.

Table 1: Overview of test instances

Company	**Employees**	**Shifts**	**Employee availability entries**	**Working time rules**
A	118	1605	1457	12
B	63	380	965	0
C	89	909	1182	9
D	101	1625	1015	2
E	51	378	473	4
F	47	835	977	7
G	19	266	440	0
H	76	513	11	1

Table 2: ALNS performance. All results are average values of 10 test runs of 900s each.

Company	**Average total iterations**	**Solution found at iteration**	**Average assigned shifts (unassigned)**	**Average solution value**	**Gap from optimal solution (%)**
A	374	365	1597 (8)	8346.27	15.23
B	68362	66548	376 (4)	1054.43	471.60
C	707	693	892 (38)	19480.56	7.86
D	797	755	1594 (31)	7924.26	N/A
E	15517	15082	374 (4)	3779.95	0.59
F	736	723	835 (0)	10458.44	2.71
G	31530	29716	266 (0)	465.03	9.28
H	8492	859	508 (0)	6437.08	0.42

5 Future developments and conclusion

The results indicate that an ALNS algorithm can be a viable alternative to MIP-solvers for the proposed model, and when computation resources are limited, it can produce results where the MIP-solver fails.

The proposed model does not address schedule costs which is an important factor to most businesses. The ideal way to do this, would be to integrate the shift assignment problem with the shift scheduling problem, as proposed in [3].

References

1. Ropke, S., Pisinger, D. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science, 40(4), 455–472 (2006)
2. Kristiansen, S., Stidsen, T. R. Elective course student sectioning at Danish high schools. Annals of Operations Research, 239(1), 99–117 (2016)
3. S. Petrovic. "You have to get wet to learn how to swim" applied to bridging the gap between research into personnel scheduling and its implementation in practice, Annals of Operations Research, 275.1, 1–19 (2019)

MISTA 2019

A Multi-Layer GP Hyper-heuristic for Real-Time Truck Dispatching at a Marine Container Terminal

Xinan Chen · Ruibin Bai · Haibo Dong

1 Introduction

Maritime transport is an important part of modern transport, while large container ports are key hubs for world trading and logistics [Christiansen et al(2007)Christiansen, Fagerholt, Nygreen, and Ronen]. With the continuous growth of container trade, the transshipment efficiency of container ports has become an important factor restricting the development of marine transportation. As the trend of develope automation terminal, resources will be automatically allocated by the dispatching system, and whether the resources can be allocated efficiently will affect the transshipment efficiency of the entire terminal significantly.

The main resources of a container port terminal can be classified into three parts: Quay Cranes, Yards and Yard Cranes, Trucks [Lorenzoni et al(2006)Lorenzoni, Ahonen, and de Alvarenga]. Quay cranes and yards are static resources which are not involved in dynamic movements. Trucks are resources that can be moved and the number of available trucks can be increased and decreased based on real needs. Therefore, whether trucks can be effectively dispatched greatly determines the operational efficiency of the entire terminal. For most port terminals, the key performance indicators is quay cranes busy rate since they need significant investment. Another indicator is the truck heavy load rate. Only loaded truck movements can be called effective movements thus our other objective is to minimize the empty truck movement.

Based on these two optimization indicators, we need to dispatch trucks in real time and arrange trucks to go to different operations points (or nodes) to complete all the

Xinan Chen
School of Computer Science, The University of Nottingham Ningbo China, Ningbo, 315100, China
E-mail: Xinan.Chen@nottingham.edu.cn

Ruibin Bai
School of Computer Science, The University of Nottingham Ningbo China, Ningbo, 315100, China
E-mail: Ruibin.Bai@nottingham.edu.cn

Haibo Dong
School of Computer Science, The University of Nottingham Ningbo China, Ningbo, 315100, China
E-mail: Nick.Dong@nottingham.edu.cn

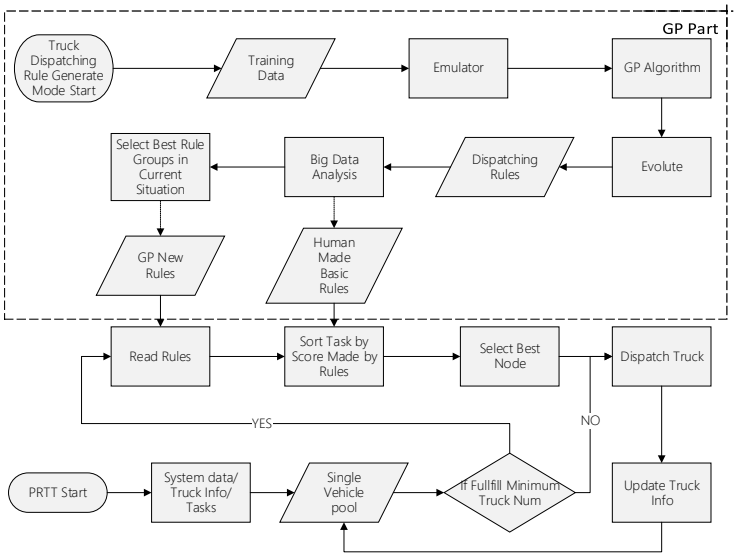


Fig. 1 Algorithm Flow Chart

required container transportation tasks between quay cranes and yard blocks (include yard to yard tasks). In the operation of the traditional container port, in order to solve the dispatching problem of trucks, trucks are usually bound to fixed quay cranes or yards, and trucks only complete tasks of bound yards or quay cranes, which is greatly limit the efficiency of the port. So, in this paper, a multi-layer GP structure will be proposed to solve the vehicle assignment problem in real time. All trucks will be aggregated into a single vehicle pool, and trucks will be dispatched based on the heuristic rules trained by GP. Each truck can freely select a task to work as its previous task is completed. This method will greatly improve the utilization rate of trucks in the port. The withdrawal or joining of a small number of trucks will no longer affect the operation of the quay crane or the yard, and the robustness of the system is improved.

2 Methodology

2.1 System Structure

In the truck dispatching model, each vehicle initially dispatched to an initial task based on the location of the vehicle and the demand of the node by the algorithm. Once the initial task is completed, the algorithm will score each available task according to the rules generated by the GP, and finally select the node with the best score. Base on the need of process and dispatch vehicles in real time, the model of the entire algorithm can be divided into two parts (shown in Figure 1), the upper part is the training part of GP, and the lower part is the actual operation part. In actual operation, the system will pass the real-time data to rules generated by the GP algorithm. Then, according to these calculation rules of the GP algorithm, the most suitable task will be assigned to each truck.

Since the complex uncertainties of port operations and the robustness required in actual operation, traditional machine learning and optimization based approaches do not perform well in the dispatching work. In practice, there are certain requirements for the readability of the algorithm. The algorithm can guarantee the reliability of the calculation only when it is highly interpretable. And the algorithm also needs to be adjusted according to the different conditions of the port. Therefore, a multi-heuristic multi-layer GP algorithm is proposed in this paper to adapt to different situations in the terminal.

2.2 GP Hyper-Heuristics

As a kind of algorithm for solving in high-dimensional space, the hyper-heuristic algorithm does not directly search for the optimal solution but searches for a more general solution in higher dimensions [Burke et al(2003)Burke, Kendall, Newall, Hart, Ross, and Schulenburg]. Therefore, it is more robust and adaptable to different problems, and can solve problems in a higher dimension which means the solution method of this algorithm may be suitable for a series of situations. There are many models of GP hyper-heuristic algorithms, such as [Bader-El-Den and Poli(2007)] [Swan and Burles(2015)] [Sim and Hart(2016)] etc.

Because its excellent generalized search capabilities, GP is an excellent carrier for hyper-heuristic. In Burke and Hyde's work [Burke et al(2003)Burke, Kendall, Newall, Hart, Ross, and Schulenburg], the problem of 2D packaging is solved by a hyper-heuristic GP algorithm. And in Zhang's paper [Zhang et al(2012)Zhang, Guan, and Xu], GP shows good ability to solve classification problems. Furthermore, Ng[Ng(2005)] has applied heuristic method to schedule crane. They both used a single-layer GP and it performed well.

2.3 Multi-Layer GP

However, in this paper, due to the complexity of the port system, the single-layer GP cannot fully address the problem. As shown in Table 1, manually designed rules are used as the benchmark. The single-layer GP's performance has been affect by the number of port trucks. It is not stable and has large fluctuations. The double-layer GP algorithm obviously has better adaptability, and it has better performance when vehicle numbers are changing (more data comparison results will be given in the presentation). The single-layer heuristic GP-inspired tree is shown in Figure 2, and multi-layer GP is a group of single-layer heuristics. Some of them are used to make decisions while others use to calculate the result. Through the calculation result, search can be projected into one dimension, which facilitates the dispatching of trucks.

3 Result Analysis & Conclusions

By using multi-layer GP, the stability is improved, indicating that the multi-layer GP can better extract the required features from the data. Since multi-layer GPs add a new heuristic layer to help select the appropriate algorithm, it cost more time in

Table 1 Performance of Different Layer Based GP Heuristic

	Artificially-Written Rule	Single Layer GP	Improve	Double Layer GP	Improve
Truck 120	0.962	0.960	-0.2%	0.963	0.1%
Truck 80	0.901	0.898	-0.2%	0.916	1.6%
Truck 40	0.646	0.658	1.8%	0.680	5.2%
Truck 20	0.398	0.385	-3.3%	0.395	-0.8%

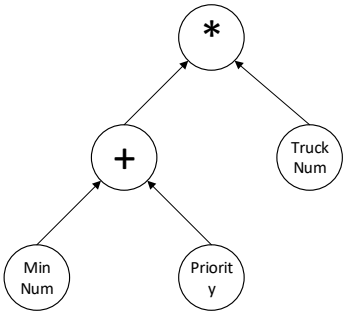


Fig. 2 Sample of single-layer Heuristic

training. And multi-layer GPs have slower convergence speeds, it may need to iterate more algebras for better results.

Unlike traditional GPs, multi-layer GPs will find answers to questions in higher abstract spaces, so they have better generalization and readability. According to the experimental results, compared with the traditional GP, the multi-layer GP has a simpler equation for each level, and has better readability. However, in the process of GP crossover and mutation, since each individual has multiple GP trees, more complex exchange and mutation logic is needed, and it is not efficient to mutate and exchange only in order.

Actually, the multi-layer GP based hyper-heuristics show better stability in the problem of assigning port vehicles. Better results were obtained across different problem instances with different sizes. Through the multi-layer GP, the upper layer algorithm is used to select the lower layer algorithm, and the training result shows better readability since heuristics are devided into different parts. This kind of structure makes the output simpler and more reliable. It perform well in truck dispatching, but performance on other classification and planning problems still needs further research.

References

[Bader-El-Den and Poli(2007)] Bader-El-Den MB, Poli R (2007) A gp-based hyper-heuristic framework for evolving 3-sat heuristics. In: GECCO, Citeseer, vol 7, pp 1749–1749

[Burke et al(2003)Burke, Kendall, Newall, Hart, Ross, and Schulenburg] Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: An emerging direction in modern search technology. In: Handbook of metaheuristics, Springer, pp 457–474

[Christiansen et al(2007)Christiansen, Fagerholt, Nygreen, and Ronen] Christiansen M, Fagerholt K, Nygreen B, Ronen D (2007) Maritime transportation. Handbooks in operations research and management science 14:189–284

[Lorenzoni et al(2006)Lorenzoni, Ahonen, and de Alvarenga] Lorenzoni LL, Ahonen H, de Alvarenga AG (2006) A multi-mode resource-constrained scheduling problem in the context of port operations. Computers & Industrial Engineering 50(1-2):55–65

[Ng(2005)] Ng W (2005) Crane scheduling in container yards with inter-crane interference. European Journal of operational research 164(1):64–78

[Sim and Hart(2016)] Sim K, Hart E (2016) A combined generative and selective hyper-heuristic for the vehicle routing problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, pp 1093–1100

[Swan and Burles(2015)] Swan J, Burles N (2015) T emplar–a framework for template-method hyper-heuristics. In: European Conference on Genetic Programming, Springer, pp 205–216

[Zhang et al(2012)Zhang, Guan, and Xu] Zhang H, Guan SU, Xu M (2012) Recursive learning of genetic algorithm featuring incremental attribute learning for higher dimensional classification problems. International Journal of Machine Learning and Computing 2(6):802

MISTA 2019

Scheduling Jobs in Flexible Flow Shops with Batching and Time Constraints

Jocelin Cailloux · Lars Mönch

1 Introduction and Problem Setting

Scheduling approaches are important in semiconductor manufacturing [6]. Time constraints that enforce a job in a wafer fab to start consecutive operations within a given time window after a certain operation is started [7] are one important class of scheduling constraints. Time constraints are established by process engineers to prevent native oxidation and contamination of the wafer surface [9]. They are often nested. Batch processing machines (BPMs) are common in wafer fabs and form another important class of constraints. BPM scheduling approaches are crucial because the processing times on BPMs are often much longer than those for operations on non-batching machines [7]. Batching decisions have therefore a large impact on the dynamic behavior of a wafer fab. Here, a batch refers to a group of jobs that are processed together at the same time on the same machine.

In the present paper, we extend the heuristics from [4] towards BPMs and the total weighted tardiness (TWT) measure. We consider a flexible flow shop with $m \geq 2$ stages where workcenter M_o on stage o consists of m_o identical parallel machines. Each job j has a due date d_j, a weight w_j, and a release date $r_j \geq 0$. The goal is to minimize the TWT value of the jobs, defined as $\sum w_j T_j$, where $T_j := \max\left(C_j - d_j, 0\right)$ for the completion time C_j of job j. Let $O_{ik}, k = 1, \ldots, m$ be the kth operation of product i and s_{jk} the start time of this operation of job j. If we have $s_{js} \leq s_{jr} + t_{irs}$ then a time constraint of length t_{irs} holds for the operations O_{ir} and O_{is} with $r < s$. The set of all time constraints is T. The products are labeled by $i = 1, \ldots, F$. Each product is defined by its route which consists of m operations. Operation O_{io} must be processed on a machine of M_o. The processing time of operation O_{io} is $p_{f(j)o}$ where f is the job-to-product mapping. The following assumptions are made:

Jocelin Cailloux
Department of Mathematics and Computer Science
University of Hagen
E-mail: jocelin.cailloux@fernuni-hagen.de

Lars Mönch
Department of Mathematics and Computer Science
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

1. BPMs might exist at stage o. In this situation each job j belongs to an incompatible job family which is determined by its product.
2. Due to the different chemical nature of the processes, only jobs of the same family can be batched together. This is typical for diffusion furnaces in wafer fabs [9,3]. Note that therefore all jobs that belong to a batch have the same processing time p_{io} when the jobs belong to product i.
3. B_o is the maximum batch size on stage o, i.e., B_o jobs can be processed simultaneously at maximum on the machines of stage o.
4. Once a machine is started, it cannot be interrupted. No preemption of the processing of a job or batch is allowed.

We have n jobs that are labeled by $i = 1, \ldots, n$. J is the set of all jobs and J_i the set of all jobs of product i. Now the problem can be represented as

$$FF_m|r_j, p-batch, incompatible, t_{irs}|TWT \tag{1}$$

using the $\alpha|\beta|\gamma$ notation from [1]. Here, the notation FF_m is used to indicate a m-stage flexible flow shop whereas $p-batch, incompatible$ refers to batching with incompatible job families on some or all stages of the flexible flow shop. While recently efficient heuristics are proposed for large-scale flexible job shops and flow shops with BPMs and the TWT measure [5,10], despite their practical importance, this is not the case for scheduling problems with BPMs, time constraints, and the TWT measure. We are only aware of a few papers for BPMs, time constraints, and the makespan measure [2, 3]. In this paper, we propose decomposition schemes that are based on mixed integer linear programming (MILP) or constraint programming (CP). Moreover, we propose a list scheduling scheme that is based on the apparent tardiness cost (ATC) dispatching rule.

2 MILP Formulation

We propose a MILP formulation for a slightly more general situation. We assume that a set of already formed batches B_{io} exists for product i on stage o. Jobs can be added to these batches if the number of jobs in a batch is smaller than B_o. The start time of batch $b \in B_{io}$ which contains $\hat{B}_{iob} < B_o$ jobs is $\hat{S}_{iob}$. The availability time of machine l on stage o is $r_{ol} \geq 0$. Newly formed batches can only be started on an available machine. The set of stages with BPMs is M^b, whereas M^1 is the set of stages without BPMs. We set $M := M^b \cup M^1$. The maximum number of batches on a stage is B. We label them by $b = 1, \ldots, B$. The following decision variables are used:

S_{obl}	:	start time of batch b on stage o on machine l
s_{jo}	:	start time of operation o of job j
x_{jobl}	:	1, if operation o of job j is scheduled in batch b on machine l, 0, otherwise
z_{job}	:	1, if operation o of job j is scheduled in the already formed batch $b \in B_{io}$, 0, otherwise
y_{iobl}	:	1, if the jobs of batch b on machine l on stage o belong to family i, 0, otherwise
u_{jol}	:	1, if operation o of job j is scheduled on machine l, 0, otherwise
e_{jko}	:	1, if operation o of job j is scheduled before operation o of job k on the same machine, 0, otherwise
T_j	:	tardiness of job j.

The MILP model is as follows where G always denotes a very large positive number:

$$\min \left\{ \omega \sum_{j=1}^{n} w_j T_j + \sum_{j=1}^{n} \sum_{o=l}^{m} s_{jo} \right\} \tag{2}$$

subject to

$$\sum_{b=1}^{B} \sum_{l=1}^{m_o} x_{jobl} + \sum_{b \in B_{io}} z_{job} = 1, \qquad j \in J, o \in M^b, i = f(j) \tag{3}$$

$$\sum_{j=1}^{n} x_{jobl} \leq B_o, \qquad o \in M^b, b = 1, \ldots, B, l \in M_o \tag{4}$$

$$\sum_{j \in J_i} z_{job} + \hat{B}_{iob} \leq B_o, \qquad o \in M^b, b \in B_{io} \tag{5}$$

$$\sum_{i=1}^{F} y_{iobl} = 1, \qquad o \in M^b, b = 1, \ldots, B, l \in M_o \tag{6}$$

$$x_{jobl} \leq y_{iobl}, \qquad j \in J, i = f(j), o \in M^b, b = 1, \ldots, B, l \in M_o \tag{7}$$

$$\sum_{l=1}^{m_o} u_{jol} = 1, \qquad j \in J, o \in M^1 \tag{8}$$

$$r_j \leq s_{j1}, \qquad j \in J \tag{9}$$

$$r_{ol} \leq s_{jo} + r_{ol}(1 - u_{jol}), \qquad j \in J, o \in M^1, l \in M_o \tag{10}$$

$$r_{ol} \leq S_{obl}, \qquad o \in M^b, b = 1, \ldots, B, l \in M_o \tag{11}$$

$$S_{obl} + \sum_{i=1}^{F} p_{io} y_{iobl} \leq S_{o,b+1,l}, \qquad o \in M^b, b = 1, \ldots, B-1, l \in M_o \tag{12}$$

$$s_{jo} + p_{io} + G(e_{jko} - 1) \leq s_{ko}, \qquad j, k \in J, i = f(j), o \in M^1 \tag{13}$$

$$s_{jo} + p_{io} \leq s_{j,o+1}, \qquad j \in J, i = f(j), o = 1, \ldots, m-1 \tag{14}$$

$$u_{jol} + u_{kol} \leq e_{jko} + e_{kjo} + 1, \qquad j, k \in J, o \in M^1, l \in M_o \tag{15}$$

$$e_{jko} + e_{kjo} \leq 1, \qquad j, k \in J, o \in M^1, l \in M_o \tag{16}$$

$$S_{obl} \leq s_{jo} + G(1 - x_{jobl}), \qquad j \in J, o \in M^b, b = 1, \ldots, B, l \in M_o \tag{17}$$

$$s_{jo} + G(x_{jobl} - 1) \leq S_{obl}, \qquad j \in J, o \in M^b, b = 1, \ldots, B, l \in M_o \tag{18}$$

$$\hat{S}_{iob} \leq s_{jo} + \hat{S}_{iob}(1 - z_{job}), \qquad j \in J, o \in M^b, i = f(j), b \in B_{io} \tag{19}$$

$$s_{jo} + G(z_{job} - 1) \leq \hat{S}_{iob}, \qquad j \in J, o \in M^b, i = f(j), b \in B_{io} \tag{20}$$

$$s_{js} \leq s_{jr} + t_{irs}, \qquad j \in J, i = f(j), (i, r, s) \in T \tag{21}$$

$$s_{jm} + p_{im} - d_j \leq T_j, \qquad j \in J, i = f(j) \tag{22}$$

$$x_{jobl}, z_{job}, y_{iobl}, u_{jol}, e_{jko} \in \{0, 1\}, \qquad j, k \in J, b = 1, \ldots, B, o \in M, l \in M^o \tag{23}$$

$$S_{obl}, s_{jo}, T_j \geq 0, \qquad j \in J, b = 1, \ldots, B, o \in M, l \in M^o. \tag{24}$$

The objective function (2) is the weighted sum of the TWT and the sum of the start times of all operations. The second term enforces that jobs are left-shifted if possible, $\omega > 0$ is a given weight. The constraints (3) ensure that each operation on a stage with BPMs is assigned to exactly one batch. Constraint sets (4) and (5) are used to respect the maximum batch size of newly formed or already existing batches. Contraints (6) make sure that only jobs of the same family belong to a batch, while constraint set (7) models that only jobs of the right family belong to a batch. It is modeled by constraint set (8) that exactly one machine belongs to each operation. Constraint sets (9), (10), and (11) make sure that the ready times of the jobs and the machine availability times are respected. The start times of batches and operations on the same stage are

represented by constraints (12) and (13). Precedence constraints among operations of the same job are modeled by constraint set (14). The order of two operations on the same machine is determined by constraints (15) and (16). Constraint sets (17) and (18) express the fact that the start times of the operations of the jobs that belong to a batch are equal to the start time of this batch. The same is expressed for already existing batches by the two constraint sets (19) and (20). Time constraints are represented by constraint set (21). Constraint set (22) linearizes the tardiness. The remaining constraints ensure that the decision variables are binary or non-negative. We also propose a corresponding CP model. However, we do not present its details due to space limitations.

3 Decomposition Heuristics

We propose a MILP-based decomposition heuristic where only at most $\bar{n} < n$ jobs are considered in each iteration. The heuristic works as follows:

1. Initialization: Sort the n jobs with respect to the global ATC dispatching rule [8]. Initialize $r_{ol} := 0$ for $o \in M, l \in M_o$ and $B_{io} := \emptyset$ for $i = 1, \ldots, F, o \in M^b$. Moreover, initialize the set S of already scheduled jobs by $S := \emptyset$ and the set of unscheduled jobs L by $L := J$.
2. Repeat for the job set $I \neq \emptyset$ containing the first $\min(L, \bar{n})$ unscheduled jobs in ATC order
   a) Schedule the jobs from I using the MILP (2)-(24).
   b) Update the r_{ol} values and the sets B_{io} based on the MILP solution obtained in Step 2a).
   c) Update the set of unscheduled jobs by $L := L \setminus I$ and the set of scheduled jobs by $S := S \cup I$.

This approach is abbreviated by MILP($\bar{n}$). Note that for $\bar{n} \leq 10$ the resulting subproblems are tractable. The approach has the advantage that nonfull batches are filled with additional jobs in future iterations if this is possible. We also propose a second decomposition scheme where we solve the subproblems by CP. It is abbreviated by CP($\bar{n}$). Due to space restrictions we are not able to provide the details of this approach.

As a reference heuristic, we propose a list scheduling approach with backtracking similar to the approach proposed in [4]. Jobs are selected based on the global ATC dispatching rule with iterative waiting time estimation [8] and scheduled one after another. Batches are incrementally filled. Infeasibilities due to the violation of time constraints are resolved by right-shifting of earlier operations of the current job until the infeasibility is removed. This procedure is abbreviated by LS-ATC. We again do not present the details due to space limitations.

4 Computational Experiments

We consider 40 problem instances that are randomly generated according to a design of experiments similar to the one used in [4]. Five stages are used for the flow shop. BPMs are on the first stage. Only $B_1 = 4$ is considered in the experiments. The number of machines per workcenter is between two and four. 20 instances have 75 jobs whereas 150 jobs belong to each of the remaining instances. Five and ten different products are considered. Up to 40 time constraints are used per instance. They can be nested.

We use $\bar{n} \in \{5, \ldots, 10\}$ in the computational experiments. The maximum computing time per subproblem is 15 seconds for $\bar{n} \in \{5, 6\}$, 30 seconds for $\bar{n} = 7$, and 60 seconds for the remaining cases. Computing times up to 19 minutes occur per instance. The computational experiments were conducted on a computer with an Intel Core i7-7700 3.6 GHz processor and 16 GB RAM using IBM ILOG CPLEX 12.7.1. We report the average TWT values obtained by the MILP- and CP-based decomposition schemes relative to th results found by the LS-ATC scheme in Table 1. We also provide the average computing time per instance. Best comparable results are marked bold.

Table 1 Computational results for the different decomposition approaches

	MILP					
$\bar{n}$	5	6	7	8	9	10
	0.8607	0.8649	0.8123	0.8157	0.7942	0.8150
time (sec)	50.2142	120.2639	322.6925	697.2149	698.0097	646.5089
	CP					
$\bar{n}$	5	6	7	8	9	10
	0.8402	**0.8344**	**0.8086**	**0.7956**	**0.7888**	**0.7799**
time (sec)	331.0875	281.8433	491.5608	874.7340	790.7478	727.7118

We see from Table 1 that both the MILP- and the CP-based decomposition heuristics perform well. Average improvements up to 22% are possible. As expected, larger values of $\bar{n}$ lead to better results. Moreover, the CP($\bar{n}$) approach slightly outperforms the MILP($\bar{n}$) approach at the expense of slightly larger computing times.

References

1. Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, 5, 287-326 (1979)
2. Ham, A., Lee, Y. H., An, J. IP-based Real-time Dispatching for Two-machine Batching Problem with Time Window Constraints. *IEEE Transactions on Automation Science and Engineering*, 8(3), 589 - 597 (2011)
3. Jung, C., Pabst, D., Ham, M., Stehli, M., Rothe, M. An Effective Problem Decomposition Method for Scheduling of Diffusion Processes Based on Mixed Integer Linear Programming. *IEEE Transactions on Semiconductor Manufacturing*, 27(3), 357-363 (2014)
4. Klemmt, A., Mönch, L. Scheduling Jobs with Time Constraints Between Consecutive Process Steps in Semiconductor Manufacturing. *Proceedings of the 2012 WinterSimulation Conference*, 2173-2182 (2012)
5. Knopp, S., Dauzére-Pèrés, S., Yugma, C. A Batch-oblivious Approach for Complex Job-Shop Scheduling Problems. *European Journal of Operational Research*, 263(1), 50-61 (2017)
6. Mönch, L., Fowler, J. W., Dauzére-Pèrés, S., Mason, S. J., Rose, O. Scheduling Semiconductor Manufacturing Operations: Problems, Solution Techniques, and Future Challenges. *Journal of Scheduling*, 14(6), 583-595 (2011)
7. Mönch, L., Fowler, J. W., Mason, S. Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Models, Analysis, and Systems. New York: Springer (2013)
8. Mönch, L., Zimmermann, J. Improving the Performance of Dispatching Rules in Semiconductor Manufacturing by Iterative Simulation. *Proceedings of the 2004 Winter Simulation Conference* 1881-1987 (2004)
9. Scholl, W., Domaschke, J. Implementation of Modeling and Simulation in Semiconductor Wafer Fabrication with Time Constraints Between Wet Etch and Furnace Operations. *IEEE Transactions on Semiconductor Manufacturing*, 13(3), 273-277 (2000)
10. Tan, Y., Mönch, L., Fowler, J. W. A Hybrid Scheduling Approach for a Two-Stage Flexible Flow Shop with Batch Processing Machines. *Journal of Scheduling*, 21(2), 209-226 (2018)

MISTA 2019

New bulky goods transportation mode in urban logistics based on Physical Internet

Luo Hao, Yang Xuan

Abstract

Recently, one of the most attractive problems in both academia and industry is how the city can behave smarter for sustainable growth. With the diverse customized demands, the urban logistics of bulky goods faces enormous challenges. Physical Internet (PI) and urban logistics are two novel concepts aiming to find a more economical and environmental way. This article aims to propose a sustainable urban logistics solution for bulky goods with the following three main contributions. First of all, we established the model of Physical Internet enabled Bulky Goods Urban Logistics (PI-BGUL). Second, the meta-heuristic algorithm (Genetic algorithm) has been applied to compare the conventional delivery mode and introduced PI-BGUL mode. Third, the presented solutions are quantitatively validated through a real-life case study in terms of economic and operational feasibility. The comparisons of PI-delivery mode and conventional delivery mode are examined under fluctuating demands and varied cost factors. The results show that the proposed PI-delivery solution can make a higher profit when the demand is stable in multiple periods.

Keywords

Physical Internet; bulky goods; urban logistics; simultaneous delivery and pick up

First Author: Luo Hao
Affiliation: Shenzhen University
E-mail: luohao403@qq.com

Second Author: Yang Xuan
Affiliation: The University of Hong Kong
E-mail: 2160050212@email.szu.edu.cn

MISTA 2019

Revisit Heuristics for Flowshop Scheduling with Availability Constraint

Yumei Huo · Hairong Zhao

1 Introduction

The flowshop scheduling model is one of the most important scheduling models that has many applications in the manufacturing industry. In this paper, we study the problem of minimizing total completion time on 2-stage flowshops. There are two machines, M_1 and M_2, and n jobs, denoted as J_i, $1 \leq i \leq n$. Each job J_i, $1 \leq i \leq n$, has two operations, a_i and b_i, that have to be processed on M_1 and M_2, respectively, in order. A machine can only process one job at a time. Each machine may have an unavailable interval during which it can't process any jobs. A job can not be preempted by another job, but may be interrupted by a machine unavailable interval. If a job is interrupted, it can be resumed after the machine becomes available. We consider the case that there is at most one unavailable interval. In the classical model, both machines are available, the problem is denoted as $F2 \mid\mid \sum C_j$. When there is an unavailable interval, we use $F2, h_{11} \mid r-a \mid \sum C_j$ and $F2, h_{21} \mid r-a \mid \sum C_j$ to denote the case that the unavailable interval occurs on M_1 and, M_2, respectively.

1.1 Literature research

This problem $F2 \mid\mid \sum C_j$ has been shown ([3]) to be NP-hard in the strong sense even if the machines are always available and there are only two machines. This is in contrast to the fact that the makespan minimization problem can be solved optimally using Johnson's rule [6]. A lot research has been conducted for this problem. This includes continuous effort in designing branch and bound algorithm to find the exact solutions since 1960s. The current best results can only solve problem size of around 50 jobs, ([1], [5]). Thus they are not practical for large instances. Given the difficulty of the problem, many researchers turn to efficient and effective heuristics. Pan and Ruiz gave a thorough

Yumei Huo
College of Staten Island, CUNY, NY 10314, USA
E-mail: yumei.huo@csi.cuny.edu

Hairong Zhao
Purdue University Northwest, Hammond, IN 46323, USA
E-mail: hairong@pnw.edu

review of the simple and composite heuristics [11]. Some of the most cited and effective ones are the RZ heuristic proposed by Rajendran and Ziegler ([12]), the NEH heuristic ([10]) which is one of the most effective heuristics for makespan objective, the B5FT that was proposed by Framinan et al. ([2]) for the total completion time problem, and LR(x) which was developed by Liu and Reeves ([8]).

With the machine availability constraint, most classical problems have to be re-considered. For flowshop scheduling with availability constraint, most research is done for two-stage flowshop problems([9]). The bulk of flow shop research has been focused on the minimization of the makespan of a schedule. For the problem of minimizing the total completion time with availability constraint, the only research is done by the authors in [7]. In that paper, the authors developed some lower bounds and dominance rules when the first machine has a single unavailable interval. A branch and bound algorithm was developed and computational experiments are performed. It is shown that up to 40 jobs can be solved.

2 New Contributions

This paper is one of the first attempts to tackle the total completion time minimization problem in flowshop model with availability constraint. We focus on the effectiveness of simple heuristics.

From [4], we know that SPT (Shortest Processing Time first) is a 2-approximation algorithm for $F2 \mid\mid \sum C_j$. For $F2, h_{11} \mid r - a \mid \sum C_j$, i.e., there is a single unavailable interval $[s, t]$ on M_1, the performance of SPT depends on the start time of the interval. It was shown in [7], that SPT can generate a schedule whose approximation ratio is n in the worst case. In this paper, we propose a modified SPT algorithm that improves the performance of SPT. Assume jobs are sorted in the increasing order of aOis, our algorithm returns the best schedule between the schedule produced by SPT and the one produced by scheduling jobs J_1, J_2, $\cdots$, J_{i_a} using SPT rule and then the remaining jobs after the unavailable interval by SPT rule, where i_a is the largest index such that $a_1 + a_2 + \cdots + a_{i_a} \leq s$.

Theorem 1 *For $F2, h_{11} \mid r - a \mid \sum C_j$ with a single unavailable interval $[s, t]$, if $min\{a_i\} \leq s < \sum_{j=1}^{n} a_j$, the modified SPT returns a schedule whose approximation ratio is $2 + \sqrt{2n}$ in the worst case.*

For $F2, h_{21} \mid r - a \mid \sum C_j$, i.e., there is a single unavailable interval on M_2, the performance of SPT also depends on the start time of the interval.

Theorem 2 *Given an instance of $F2, h_{21} \mid r - a, \mid \sum C_j$ with unavailable interval $[s, t]$. Let S^* be the optimal schedule of the instance. Let $C_{\max}(SPT)$ be the makespan of the SPT schedule of the instance. $C^*_{\max}$ be the optimal makespan of the instance obtained by removing the unavailable interval. Then*

1. *if $s > C_{\max}(SPT)$ or $t \leq \min\{a_j\}$, then $\sum C_j(SPT) \leq 2 \sum C_j(S^*)$*
2. *if $C^*_{\max} < s < C_{\max}(SPT)$, then $\sum C_j(SPT)$ may be arbitrarily large.*
3. *if $s < C^*_{\max}$, then $\sum C_j(SPT) \leq (2 + \frac{n+1}{2}) \sum C_j(S^*)$*

We also revisit some effective heuristics that were proposed for $F2 \mid\mid \sum C_j$. We investigated their performance for $F2, h_{11} \mid r - a \mid \sum C_j$ and $F2, h_{21} \mid r - a \mid \sum C_j$.

We generated 2000 instances that differ in number of jobs, distribution of processing times, start and duration of the unavailable interval. Among theses instances, there are 120 instances for problem $F2 \mid\mid \sum C_j$, 1440 instances for each of $F2, h_{11} \mid r-a \mid \sum C_j$ and $F2, h_{21} \mid r-a \mid \sum C_j$.

Through extensive computation experiments of 39 heuristics, we have the following observations.

1. Almost all heuristics perform the best when both machines are available, and performs slightly worse when the unavailable interval is on M_1 than the case when the unavailable interval is on M_2.
2. Effective heuristics such as LR(X) for the classic model still perform quite well when there is availability constraints. Although SPT can be very bad theoretically, in practice, it is a very effective heuristic.
3. If the unavailable interval happens on M_1, the heuristics performance worsens as n increase; but if the unavailable interval happens on M_2, the performance improves as n increases.
4. The performance of each heuristic only increases a little as the processing time increases from $[1, 10]$ to $[1, 50]$. This is in contrast with the branch and bound method, whose performance highly correlates with the processing time. In terms of the start time, the performance of a heuristic is relatively stable, and slightly increases as the duration of the unavailable interval increases.

References

1. Akkan, C. and S. Karabati, The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm, European Journal of Operational Research, 159, 420-429, 2004.
2. Framinan JM, Leisten R, Rajendran C. Different initial sequence for the heursitics of Nawaz, Enscore and Ham to minimize makespan, ideltime or flowtime in the static permutation flowhsop. International Journal of Production Research 2003, 41: 121-148.
3. Garey, M. R., D. S. Johnson, and R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research, 13, 330-348, 1976.
4. T. Gonzalez and S. Sahni, Flowshop and jobshop schedules: Complexity and approximation. Operations Research 26, 36-52.
5. Hoogeveen, J. A., L. van Norden and S. L. Van de Velde, Lower bounds for minimizing total completion time in a two-machine flow shop, Journal of Scheduling, 9: 559-568, 2006.
6. Johnson, S. M., Optimal Two- and Three-Stage Production Schedules With Set-up Time Included, Naval Research Logistics Quarterly. 1: 61–68, 1954.
7. Huo, Y. and H. Zhao, Minimizing Total Completion Time in Flow shop with Availability Constraints, 9th International Workshop on Computational Optimization (WCO'16), Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS 2016), page 637-645.
8. J.Y. Liu and C. R. Reeves, Constructive and composite heuristic solutions to the $P \mid\mid \sum C_i$ scheduling problem, European Journal of Operational Research, 132, pp. 439-452, 2001.
9. Ma, Y., C. Chu and C. Zuo, A survey of scheduling with deterministic machine availability constraints, *Computers & Industrial Engineering*, 58(2), pp. 199-211, 2010,
10. M. Nawaz, E.E. Enscore Jr, I. Ham, A heuristic algorithm for the m-machine, n-job flowshop sequencing problem Omega, 11 (1) (1983), pp. 91-95.
11. Q.-K. Pan, R. Ruiz, A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime, Computers & Operations Research 40 (2013) 117–128
12. Rajendran C, Ziegler H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research 1997;103(1):129–38

MISTA 2019

Coordinating Resource Sharing for Joint Optimization among Carrier Alliances in Liner Shipping (Abstract)

Zhou Xu · Xiaofan Lai · Liming Liu

In this work, we propose a new approach to analyzing the coordination of resource sharing for carrier alliances. It extends a coalition game theory of a market with transferable payoff known from exchange economies [1,3]. We utilize a competitive equilibrium of the market to show that under a general setting, exchange prices for coordination always exist, and can be obtained by using convex optimization methods [2]. Such exchange prices can coordinate carriers, even when their selfish behaviors are of different types and are only partially known, and hence having significant practical values. With our new approach applied, we (i) unify and extend the existing results by simpler proofs, (ii) correct a faulty analysis and address an open question appeared in the literature, and (iii) derive exchange prices for coordination of carriers' resource sharing under several new and complicated settings that reflect practical concerns but have never been studied before, such as those with Nash equilibrium constraints and those with uncertainties in cargo demands. Besides, our study has also (iv) bridged the theories in exchange economies, the solution methods in convex optimization, and the applications in coordination of resource sharing for carrier alliances.

Acknowledgements This research is supported in part by the National Natural Science Foundation of China (Grant no. 71401149) and Hong Kong Polytechnic University (Grant no. 4-BCCP).

References

1. G. Debreu and H. Scarf. A limit theorem on the core of an economy. *International Economic Review*, 4(3):235–246, 1963.
2. D. G. Luenberger and Y. Ye. *Linear and nonlinear programming*, chapter 11. Constrained Minimization Conditions, pages 321–358. 2008.
3. M. Osborne and A. Rubinstein. *A Course in Game Theory*, chapter 13. The Core, pages 258–275. The MIT Press, 1994.

Zhou Xu
Hong Kong Polytechnic University, Hong Kong SAR, P.R. China
E-mail: lgtzx@polyu.edu.hk

Xiaofan Lai
Sun Yat-sen University, Guangzhou, P.R. China

Liming Liu
Lingnan University, Hong Kong SAR, P.R. China

MISTA 2019

A Branch-and-Price Algorithm for the Electric Vehicle Routing Problem with Nonlinear Charging Function

Zhixing Luo

In this paper, we propose a branch-and-price algorithm for the electric vehicle routing problem with nonlinear charging function (EVRP-NCF), a new VRP variant introduced to the literature recently. Although the electric vehicle routing problem (EVRP) has been a hot research topic in operations research, most of the existing EVRPs have an unrealistic assumption that the charging speeds of the charging stations are constant. The EVRP-NCF relaxes this assumption and defines the charging functions to be concave piece-wise functions. Until now, no exact algorithms have been proposed to solve the EVRP-NCF. Our branch-and-price algorithm is the first exact approach for the EVRP-NCF with a set of new dominance rules in the label-setting algorithm for solving the pricing problem. In opposite to the traditional dominance rules, ours consider a set of labels to dominate a label, hence significantly increasing the chance for a successful dominance. Computational results on the benchmark instances demonstrate the effectiveness and efficiency of our approach.

Zhixing Luo
Nanjing University
E-mail: luozx@nju.edu.cn

MISTA 2019

Routing Optimization with Generalized Consistency Requirements

Jun Xia

The goal of existing consistent vehicle routing problems (ConVRPs) aims to design synchronized routes on multiple days to serve a group of customers while minimizing the total travel cost. It stipulates that customers should be visited at roughly the same time (time consistency) by several familiar drivers (driver consistency). This paper generalizes ConVRPs for any level of the driver consistency and addresses route consistency, meaning that each driver is confined to a small set of routes, expecting to improve efficiency by increasing route familiarity. To solve this problem, we develop two set-partitioning-based formulations, one based on routes and the other based on schedules. We investigate valid lower bounds on the linear relaxations of both formulations. They are utilized to derive a subset of columns (routes and schedules), for which the schedule subset is constructed by the route subset; within the subset are columns of an optimal solution for each formulation. We then solve the reduced problem of either formulation to achieve an optimal solution. Numerical results show that our exact method can effectively solve most of the medium-sized ConVRP instances in the literature, and also solves some newly generated instances involving up to 50 customers following a general consistency setting. Our exact solutions explore several interesting insights: First, maintaining the consistency requirements at the highest levels is unreasonably costly, but maintaining high levels may not. Second, high time consistency levels could be awarded by imposing other two consistency requirements. Third, traditional consistency requirements do not necessarily improve service efficiency.

Jun Xia
Sino-US Global Logistics Institute, Shanghai Jiao Tong University
E-mail: lgtxiaj@sjtu.edu.cn

Author Index

Ada C. 224
Aickelin U. 16
An H. 537
Aronshtam L. 30
Bai D. 109
Bai Q. 360
Bai R. 16, 156, 576, 598
Baruch M. 65, 293
Bassimir B. 381
Biyanto T. R. 340
Böðvarsdóttir E. B. 141
Bouznif M. 42
Braune R. 591
Cailloux J. 603
Çalık H. 186
Chan H. K. 463
Chau V. 558
Chen K. 445
Chen P. 169
Chen Q. 320, 350
Chen S-M. 549
Chen T.K. 10
Chen X. 598
Chen Y. 499, 507
Chen Z-L. 546
Christensen S. K. 595
Cui L. 360
Cui T. 156
Dana S. 65, 293
Davari M. 503

De Causmaecker P. 53
Deng P. 271
Deng Q. 257
Ding H. 328
Ding S. 156
Dong H. 598
Fan G-Q. 572
Fang K. 100
Fang W. 573
Feng S. 249, 558
Fränti P. 584
Fuchigami H. Y. 396
Gao L. 555, 557
Ge P. 16
Gong H. 564
Goossens D. 301, 495
Grigoreva N. 487
Grinshpoun T. 125
Guan Z. 567, 573
Guo X. 541, 556
Hall N. G. 11
Hallaert A. 186
He P. 553
Hermans B. 481
Hu Y. 76
Huang L. 445
Huang Y. 285, 563
Huo Y. 609
Ikar E. 125
Ikegami A. 484
Ilani H. 30, 125
Itoi M. 484
Ji M. 285

Jin H. 510
Jin M. 100
Jouglet A. 42
Kang L. 537
Karami F. 186, 197
Kusuma D. W. 340
Kyngäs N. 301
Lai X. 612
Lee J. 537
Leus R. 481, 503
Li F. 546, 580
Li K. 360, 533, 553
Li M. 558
Li P. 557
Li X. 224, 555, 556, 557
Li Y. 554, 556
Li Z. 320, 350
Lin L. 507
Lin W-C. 109
Lin Y. 76
Liu J. 320, 350
Liu L. 612
Liu P. 550
Liu S. 510, 583
Liu Z. 271, 470, 572
Lu L. 554
Lu X. 550
Lu X-Q. 429
Lu Z. 16
Luo D. 567
Luo H. 608
Luo W. 100
Luo Z. 613

Matuschke J. 481
Moench L. 603
Nace D. 42
Nenonen H. 584
Nissen V. 209
Nugroho H. 340
Nurit B. 65
Nurmi K. 301
Oliveira J. F. 12
Pan W. 583
Park S. 537
Peng C. 320
Peng J. 285
Pham T-S. 53
Pi B. 499
Pietrowicz S. 42
Qingcheng Z. 570
Qu C. 445
Qu R. 576
Rousseau L-M. 53
Santos H. G. 410
Schulte J. 209
Shan L. 570
Shi F. 565
Shufan E. 30, 125
Smet P. 141
So R. 510
Song Y. 499
Stidsen T. 141, 500
Su L. 563
Sun D. 561
Sun X. 533
Tan J. 169

Tan K. 507
Tan Z. 507
Tang L. 13, 546, 561, 563, 564, 565, 571
Tang S. 470
Thanos E. 410
Tian Y. 271
Tu C. 16
Vacher B. 42
Van den Bossche T. 410
Vancroonenburg W. 197, 410
Vanden Berghe G. 141, 186, 197, 410
Wan L. 271
Wang G. 557, 571
Wang H. 530
Wang J-Q. 572
Wang L. 14
Wang S. 580
Wang Y. 257, 558
Wanka R. 381
Wei W. 503
Wu C-C. 109
Wu C-X.. 429
Wu W. 76, 484
Xiang W. 285
Xia J. 614
Xie J. 555
Xing L. 499
Xiong Q. 249
Xiong T. 271
Xu D. 360
Xu M. 169
Xu S. 463
Xu Z. 612

Yagiura M. 76
Yang A. 566
Yang D. 580
Yang F. 503
Yang Q. 470
Yang X. 608
Ye C. 445
Yi X. 495
Yin Z. 571
Yu Y. 328
Yue L. 530, 573
Yue Q. 491
Zeng Q. 566
Zeng Q-C. 549
Zhang G. 558
Zhang L. 554
Zhang M-X. 429
Zhang S. 169
Zhang X. 109, 285
Zhang Y. 558, 576
Zhang Z. 499
Zhao H. 608
Zhao J. 16
Zhao T. 445
Zhao X. 249
Zheng Y-J. 429
Zhong H. 350
Zhong W. 328
Zhou S. 491
Zhou Y. 541