The stochastic container relocation problem with flexible service policies

Yuanjun Feng [a], *Dong-Ping Song* [a,*], *Dong Li* [a]

[a] School of Management, University of Liverpool, Chatham Street, Liverpool, L69 7ZH, UK

Qingcheng Zeng [b]

[b] School of Maritime Economics and Management, Dalian Maritime University, 1 Linghai Road, Dalian, 116026, China

Abstract: This paper investigates the Stochastic Container Relocation Problem in which a flexible service policy is adopted in the import container retrieval process. The flexible policy allows the terminal operators to determine the container retrieval sequence to some extent, which provides more opportunity for reducing the number of relocations and the truck waiting times. A more general probabilistic model that captures customers' arrival preference is presented to describe the randomness for external truck arrivals within their appointed time windows. Being a multi-stage stochastic sequential decision-making problem, it is first formulated into a stochastic dynamic programming (SDP) model to minimize the expected number of relocations. Then, the SDP model is extended considering a secondary objective representing the truck waiting times. Tree search-based algorithms are adapted to solve the two models to their optimality. Heuristic algorithms are designed to seek high-quality solutions efficiently for larger problems. A discrete-event simulation model is developed to evaluate the optimal solutions and the heuristic solutions respectively on two performance metrics. Extensive computational experiments are performed based on instances from literature to verify the effectiveness of the proposed models and algorithms.

Keywords: stochastic container relocation problem, appointment time window, flexible service, stochastic dynamic programming, tree search-based algorithm

1 Introduction

As critical nodes in the global container transport networks, container terminals play an important role in transhipping containerized cargoes between different transport modes. At container terminals, containers are handled through a series of operations, which can be generally divided into seaside operations (unloading/loading operations) and landside operations (stacking/retrieval operations). Methods for improving the operational efficiencies at container terminals have been studied for years and many models and algorithms have been developed (see review papers: Stahlbock and Voß, 2008; Zhen et al., 2013; Lehnfeld and Knust, 2014; Carlo et al., 2014; Lee and Song, 2017; Dragović et al., 2017).

One major source of inefficiency in most container terminals is the relocation move (Caserta et al., 2011a; Ku and Arthanari, 2016a). In a typical container terminal, containers are stored in the terminal yard after their arrivals, waiting for onward transport. The storage area of a yard is divided into blocks, each including 20-40 bays with each bay consisting of several stacks. Containers are often piled up vertically in stacks. During the container retrieval process, if the target container to be retrieved is not on the topmost tier, those above it – that is, the *blocking* containers - need to be moved to other stacks in the same bay. The move of blocking containers is called *relocation*, *reshuffling*, or *rehandling*, which is an unproductive operation. The Container Relocation Problem (CRP) aims at seeking a sequence of moves to retrieve all containers from a given bay with the minimum number of relocations, which is a combinatorial optimization problem. Most existing studies on the CRP assume a priori given container retrieval sequence. The CRP for import containers whose retrieval times are subject to uncertainty has been less investigated.

For import containers, the stochastic arrival times of external trucks complicates the CRP as it will easily result in re-relocations in the future. The Truck Appointment System (TAS), also known as vehicle booking system (VBS), can

* Corresponding author.

E-mail addresses: Yuanjun.Feng@liverpool.ac.uk (Y. Feng), Dongping.Song@liverpool.ac.uk (D. Song), Dongli@liverpool.ac.uk (D. Li), qzeng@dlmu.edu.cn (Q. Zeng).

increase the predictability for truck arrival times, which has been implemented in many container ports to control the truck arrivals at the terminal (Davies, 2009). Under TAS, a truck must make an appointment with the terminal in advance to indicate a time window in which the truck will arrive at the terminal. Therefore, each arrival truck will have an appointed arrival time window. As a result, the arrival precedence of trucks with different appointed time windows becomes pre-specified. However, the arrival sequence of the trucks within the same appointed time window remains uncertain, which is typically revealed during the retrieval process. The CRP that considers the randomness truck arrivals in the same time window is termed as the Stochastic Container Relocation Problem (SCRP) or the CRP with Time Windows (CRPTW) in the literature.

The SCRP is more realistic to model the import container retrieval process. Among the very few studies on the SCRP (e.g. Ku and Arthanari, 2016a; Galle et al., 2018b), a common assumption is that retrieval requests are fulfilled on a first-come-first-served (FCFS) basis. The FCFS rule appears to be reasonable in practice to ensure service equity but may lead to a sub-optimal solution from the overall system perspective. Besides, the service equity of the FCFS rule is debatable, because the trucks may experience different waiting times and the required number of relocations may be affected by previous trucks. Truck waiting time is part of the truck turn time, which is a key performance metric to measure the efficiency of a container terminal and also contributes to the evaluation of customer service levels and port competitiveness (de Melo da Silva et al., 2018). Some ports (e.g., Port Botany; Port Metro Vancouver) are even charged for a penalty if they exceed a stipulated turn time. As an alternative to the FCFS service, a flexible service may yield more opportunities for optimization on the number of relocations, as well as the truck waiting time. In this paper, we extend the SCRP to a general setting that allows some flexibility in the container retrieval sequences. We term this type of problem as the *SCRP with flexible service policies* or *SCRP-FS*.

With the SCRP-FS, we also generalize the probabilistic model of truck arrivals. In the existing studies, the arrival order of trucks booked in the same time window is assumed to be uniformly distributed, which is not necessarily realistic. Customers (truckers) may have different preferences for different segments of their appointed time windows. Firstly, in the TAS, the trucks may not always get their desired time windows because slots are often oversubscribed (Mongelluzzo, 2019). In order to narrow the deviation from its desired time window, the truck will have preference for either the earlier segment or the latter segment of the shifted appointed time window. Secondly, some terminal operators (e.g., DP World, Patrick) impose financial penalties on no-show (or late) trucks to ensure truckers compliance with their appointed time windows. To avoid such penalty, those trucks that are subject to high uncertainties on the road tend to target the earlier segment of the appointed time window. Such customer preference information may come from the TAS or from historical data that record the trucks' arrival behavior, which can be utilized to make better decisions.

This paper aims to investigate how to use flexible service policies to improve import container relocation and retrieval in the presence of truck arrival uncertainties characterized by customers' preference. The objective of this paper is: (i) to seek the optimal solution (including retrieval sequence and relocation positions) that retrieves all containers from a given bay considering both terminal relocation and external truck waiting; (ii) to quantify the reduction on the number of relocations and the truck waiting times by adopting the flexible service policy; (iii) to evaluate the impact of different bay layouts (i.e. size and fill rate) and truck arrival patterns (i.e., the number of trucks booked in a time window and the customer preference) on the effectiveness of the flexible policy compared with the FCFS rule; (iv) and to analyze the influence of customer preference on the results.

Our contributions to the existing literature and practice can be summarized as follows: (i) We propose a new service policy to improve import container retrieval performance in the context of stochastic container relocation problem (SCRP) and generalize the SCRP to be SCRP-FS, which can provide more opportunities for optimizing the current retrieval practice. We also generalize the probabilistic model of truck arrivals so that the customers' preference-based arrival behavior can be captured more accurately. (ii) We introduce a new optimization framework that jointly optimizes the expected number of relocations and the truck waiting times. The proposed model enables port operators to quantify the benefits of controlling the truck service sequence to both terminals and truckers, which is the first in the SCRP studies.

(iii) We develop exact algorithms by extending an existing algorithm with major adaptions. The incremental contributions of our exact algorithms include a decision tree with a new structure, a new lower bound for the expected number of relocations for the SCRP-FS, and a procedure for minimizing the truck waiting times batch by batch. We develop two efficient heuristic algorithms for solving the SCRP-FS, which can serve as decision support tools for terminal operators in real applications. (iv) We construct a discrete-event simulation model for evaluating the exact solutions in terms of two performance measures: the expected number of relocations and the truck waiting time. To the best of our knowledge, this study is among the first to evaluate the truck waiting time of the exact solutions with a tree structure in the context of uncertain CRP. (v) We conduct extensive experiments to demonstrate the effectiveness of the flexible service policy and the influence of the customer preference on the results. The results can provide managerial insights for terminal operators to manage import container operations more efficiently.

The remainder of the paper is organized as follows. Section 2 reviews the previous work related to the CRP and SCRP and discusses the service policies applied in the (S)CRP. Section 3 describes the problem in detail and formulate it using stochastic dynamic programming. Section 4 and section 5 respectively propose exact solution algorithms and heuristic solution methods. A simulation model is developed in Section 6 to evaluate the solutions. The results of computational experiments are reported in Section 7. Section 8 summarizes the findings and provides managerial insights.

2 Literature review

Container relocation is related to several container handling processes at container terminals. Four types of relevant problems have been identified by Carlo et al. (2014): storage location assignment problem, joint retrieval sequencing and relocation problem, pre-marshalling problem and re-marshalling problem. An overview of these problems from a mathematical perspective is given in Lehnfeld and Knust (2014). The joint retrieval sequencing and relocation problem is the focus of this paper.

2.1 Deterministic CRP and uncertain CRP

Most studies on the joint retrieval sequencing and relocation problem assume a priori given retrieval sequence and only focus on optimizing relocation positions, which leads to the standard CRP. The basic objective of the standard CRP is to retrieve all containers in a given bay in a pre-defined order with the minimum number of relocations. There are also several variants of the CRP, such as the dynamic CRP, the unrestricted CRP, etc. We classify the relevant literature into two research streams: *deterministic CRP* and *uncertain* CRP, which are differentiated by whether the information on the containers' retrieval times/sequences is deterministic or uncertain.

Previous researches have largely concentrated on the first research stream, i.e. the certain version of the CRP. A number of (mixed) integer programming have been proposed to solve the problem (e.g. Wan et al., 2009; Caserta et al., 2012; Petering and Hussein, 2013; Tang et al., 2015; Zehendner et al., 2015; Expósito-Izquierdo et al., 2015; Galle et al., 2018a). Other studies focus on developing effective solution algorithms. Exact solution algorithms are mainly by search-based algorithms, e.g., (iterative deepening) A* algorithms (Zhu et al., 2012; Borjian et al., 2015a; Quispe et al., 2018), Branch and Bound (B&B) (Kim and Hong, 2006; Ünlüyurt and Aydın, 2012; Expósito-Izquierdo et al., 2015; Tanaka and Takii, 2016), Branch and Price (Zehendner and Feillet, 2014), and the abstraction method (Ku and Arthanari, 2016b), Besides, heuristics algorithms are presented to overcome computational complexities of the CRP, e.g., beam search algorithms (Bacci et al., 2019; Ting and Wu, 2017) and greedy heuristics (Jin et al., 2015; Jovanovic and Voß, 2014) (c.f.Ku and Arthanari, 2016b and the references therein).

In the second research stream, the uncertain CRP may be further categorized into two sub-groups: the online setting and the probabilistic setting, according to whether the uncertainties of the containers' retrieval times/sequences are represented by probabilities or not.

In the online setting, the knowledge of the exact container retrieval sequence is limited to a given look-ahead horizon and is revealed over time gradually, and the research focus is to design efficient online heuristics to relocate containers in real-time. Zehendner et al. (2017) investigate a case of the online CRP where the look-ahead horizon is zero and one container is revealed at a time. They analyze the theoretical performance of an online relocation rule called heuristic *L*

(leveling) that relocates containers to the lowest tier. Zhao and Goodchild (2010) make use of truck appointment information to deal with the online CRP using a simulation method. At the beginning of the retrieval process, the retrieval containers booked in different time windows are known, but the exact retrieval sequence of the containers booked in the same time windows is unknown or partially known, which is revealed as the retrieval proceeds periodically. Two heuristics are designed to reduce the number of relocations utilizing the truck arrival information.

In the probabilistic setting, the uncertainties on the containers' retrieval times/sequences are modeled by a probability distribution and the research purpose is to minimize the expectation of the performance measures. Given the probabilistic distribution of containers dwell times, Tong et al. (2015) propose two heuristic rules to determine the positions of relocated containers with the objective of minimizing the total expected number of relocations for retrieving all the containers from a bay. Considering groups of containers with uncertain group retrieval orders, de Melo da Silva et al. (2018) introduce the Block Retrieval Problem (BRP) and the Bi-objective Block Retrieval Problem (2BRTP). The BRP aims to minimize the number of relocations for the initial target group by optimizing the retrieval sequence and the relocation positions, which is solved by a B&B algorithm and a linear time algorithm. Then, assuming that the probability of any remaining group being the forthcoming one is known, the 2BRTP is introduced with the primary objective of minimizing the number of relocations for the initial target group and the secondary objective of minimizing the expected number of relocations for the forthcoming group. A B&B algorithm and a beam search algorithm are proposed for solving the 2BRTP.

The above studies of the uncertain CRP focus on the solution algorithms without providing the details of problem formulation. Mathematical optimization models for the uncertain CRP in the probabilistic setting are presented in a few studies. Borjian et al. (2013) introduce a two-stage stochastic optimization framework for the CRP with partial information, in which the departure times of some containers are known, while for the remaining containers only a probability distribution of the retrieval order is given. The model is to minimize the weighted sum of the expected number of relocations and total retrieval delays, which yields the optimal sequence of moves for each possible scenario. A heuristic based on the stochastic optimization model is designed to obtain sub-optimal solutions. Ku and Arthanari (2016a) propose the CRP with Time Windows (CRPTW), in which the retrieval sequences of containers with different departure time windows are in ascending order by their departure time windows. It is assumed that containers with the same departure time windows are retrieved in the uniformly distributed order which is revealed one container at a time. The problem is formulated into a stochastic dynamic programming (SDP) model to minimize the expected number of relocations. A search-based algorithm (depth-first search) in a tree space is proposed to solve the model optimally. More recently, Galle et al. (2018b) study a similar CRPTW using the term SCRP. Different from Ku and Arthanari (2016a), the full exact retrieval order of containers booked in the same time window is revealed at once after all containers booked in the previous time window have been retrieved. The SCRP is formulated as a multi-stage stochastic model, called the batch model. An optimal search-based algorithm called Pruning-Best-First-Search (PBFS), a randomized approximate algorithm called PBFSA, and two new heuristics are proposed to solve the batch model. The batch model is compared with the SDP model in Ku and Arthanari (2016a) both theoretically and computationally to prove that it is beneficial to use the batch model in terms of the expected number of relocations. Note that because of the use of the same information revealing mechanism and the similar modelling techniques to seek global optimal solutions, the current paper positions itself to the SCRP proposed by Galle et al. (2018b). However, our study differs from Galle et al. (2018b) in many aspects that will be elaborated at the end of this section.

Finally, it is worth mentioning that several interesting variants of the (S)CRP have also been investigated. For instance, a few studies consider the CRP in a three-dimensional storage area and take into account both the number of container movements and the working time of the yard crane (e.g., Lee and Lee, 2010; Lin et al., 2015). From the service-oriented standpoint, López-Plata et al. (2017) address the Blocks Relocation Problem with Waiting Times (BRP-WT) focusing on minimizing the sum of waiting times of a set of blocks during retrieval. Recently, Zweers et al. (2020) present a new optimization problem related to the SCRP, called the Stochastic Container Relocation Problem with Pre-Processing (SCRPPP), which aims to minimize a weighted average of the pre-processing moves in the pre-processing phase (when

the crane is idle) and the relocation moves in the relocation phase. A B&B algorithm and a local search heuristic are proposed to solve the problem.

2.2 Service policies

Among the literature reviewed above, most of them assume that each container has a distinct retrieval sequence that is exogenously determined. For import containers, this assumption corresponds to the FCFS service policy, under which the containers are retrieved in the order of external truck arrivals. Using the FCFS service policy may make the problem more tractable but meanwhile loses some opportunities for optimization. In the uncertain CRP context, three studies have considered flexible service policies or service out-of-order (SOOO). Zhao and Goodchild (2010) propose a pickup sequence dictation heuristic that dictates the retrieval sequence of containers within the first arrival group to reduce relocations. In the study of de Melo da Silva et al. (2018), the containers in the same group are assigned the same retrieval priority and their retrieval order is to be optimized. Borjian et al. (2013) assume a service time window for each container instead of imposing strict service order on containers. In the deterministic CRP context, a few researchers have also introduced the flexibility of container retrieval sequence, e.g., Kim and Hong (2006), Borjian et al. (2015b) and Zeng et al. (2019).

One major concern of flexible retrievals is the possibility of causing extra delay and service inequity to some trucks. Zhao and Goodchild (2010) and Zeng et al. (2019) attempt to avoid this by restricting out-of-order retrievals within a group of containers booked in the same time window. However, Zhao and Goodchild (2010) do not analyze the impact of dictating the pickup sequence on external trucks. Zeng et al. (2019) find that when the number of containers booked in each time window is over a certain number, adjusting the pickup sequence may increase the average waiting time of external trucks. One possible reason for this is that they do not consider the common phenomenon of trucks queuing before getting the retrieval service at congested container terminals, where out-of-order retrievals may not create more waiting times. Borjian et al. (2015b) control the level of flexibility by limiting the number of out-of-order retrievals before each truck. They conclude that the average retrieval delay is decreased as a result of out-of-order retrievals, and in the long term, the service equity that each truck receives is not adversely affected. In the study of Kim and Hong (2006), the containers in the same group are assumed to be loaded into a cluster of slots of a vessel in any order, which means those containers share the same retrieval priority, and thus there is no issue of delay and service inequity. Similarly, de Melo da Silva et al. (2018) do not consider this issue either, as they assume that the containers in the same group are to be retrieved by the same customer. Borjian et al. (2013) set a maximum service delay for each container and consider a weighted objective function that jointly minimizes the expected number of relocations and total delays. It can be seen that Borjian et al (2013) is the only paper that applies flexible retrievals in an uncertain CRP using a mathematical optimization model. However, in their model, all uncertain information is revealed at once, which is not close to reality.

In this paper, out-of-order retrievals are limited to the trucks arriving in the same sub-time window to maintain the service equity among subsystems (Yang et al., 2013) and to avoid excessive delay to any trucks. Meanwhile, this ensures that trucks arriving in the first sub time window are served before those arriving in the second sub-window, which is consistent with the customers' preferences.

Table 1 compares this paper with the closely related studies from four key aspects. This paper is differentiated from previous works in several ways. From the problem perspective, the SCRP-FS we propose generalizes the SCRP in the sense that first, a flexible service policy (SOOO), as opposed to the FCFS policy, is integrated into the multi-stage stochastic optimization framework. The SOOO policy allows some flexibility in the retrieval sequences of containers in the same group and thus provides more opportunities to reduce the number of relocations and the truck waiting times. Second, instead of assuming uniformly distributed truck arrival order, we propose a more general probabilistic model to describe the randomness of the truck arrivals within the same group. Specifically, our probabilistic model has the capability of capturing the customer preference-based arrival behavior, which has more practical relevance. From the methodology perspective, we propose a new optimization framework that not only optimizes the expected number of relocations (primary objective) but also optimizes the truck waiting times (secondary objective). Although our exact

solution algorithm is built upon the PBFS algorithm in Galle et al. (2018b), the existing PBFS algorithm does not allow for a straightforward adaption to our problem due to the substantial differences between the SCRP and the SCRP-FS. A great deal of effort has been made to adapt the PBFS algorithm to solve our problem. The major adaptions are: first, we construct a more general decision tree with a new structure, which adds a new layer of decision node for sequencing trucks and expresses nodes with a dual-matrix configuration that represents both truck appointment information and customer preference; second, we propose a new lower bound for the expected number of relocations for the SCRP-FS by including the characteristics of flexible retrieval orders and customer preference-based arrivals, which is necessary to prune unpromising nodes; third, we add a procedure for minimizing the truck waiting times batch by batch by using the derivation of a waiting time indicator. In addition, we design two fast and efficient heuristic algorithms for the SCRP-FS. Last, we construct a discrete event simulation model to evaluate the exact solutions with a tree structure, which is the pioneer in the relevant literature. The simulation model is especially needed to evaluate the truck waiting times for the exact solutions because the exact algorithms do not record time-related performance.

Table 1 The comparison with the most relevant studies

Characteristics	The probabilistic model of truck arrival	Information updating	Service policy	Objectives
Borjian et al. (2015b)	Deterministic	-	A limited number of out-of-order retrievals before each truck	The weighted number of relocations and retrieval delays
Zeng et al. (2019)	Deterministic	-	Out-of-order retrieval within each group	The number of relocations
Borjian et al. (2013)	Scenario-based uncertainty	Two-stage	Out-of-order retrievals s.t. a maximum delay	The weighted expected number of relocations and total delays
Ku and Arthanari (2016a)	Uniform distribution	Multi-stage over individual trucks	FCFS	The expected number of relocations
Galle et al. (2018b)	Uniform distribution	Multi-stage over groups	FCFS	The expected number of relocations
This paper	Customer preference-based uncertainty (incl. uniform distribution)	Multi-stage over groups	Out-of-order retrievals within each sub-group	Primary objective: the expected number of relocations; Secondary objective: the total truck waiting times of each group

3. Problem description and formulation

In this section, we first describe the SCRP-FS in detail and then formulate the problem by stochastic dynamic programming.

3.1 Problem description

The studied problem is a multi-stage stochastic optimization problem. The problem is described along with the introduction of the basic assumptions of the SCRP, the probabilistic model of truck arrivals, the containers' attributes, and the service priority.

3.1.1 Basic assumptions

The following assumptions are generic to the (S)CRP (e.g. Kim and Hong, 2006; Caserta et al., 2011b; Ku and Arthanari 2016a; Galle et al., 2018b).

A1: Relocations are performed only within the bay being considered. The initial bay layout consisting of S stacks, T tiers, and C containers. In order to avoid infeasible relocations, the storage capacity of the bay is restricted to be $(S-1)T+1$ containers.

A2: A container is relocatable only when it is blocking the target container.

A3: No new containers arrive at the bay during the container retrieval process.

A4: The travel distance of the trolley and spreader of the yard crane does not have an impact on relocation costs, which means that the relocation effort is measured only by the number of relocations.

A5: Each container is booked to a time window and the corresponding truck will arrive at the terminal within the

appointed time window. A ***batch*** of containers (trucks) (i.e., one container corresponds to one truck) is defined as the set of containers (trucks) booked to the same time window. The arrival precedence relationship among batches of trucks is known, but the exact arrival order of trucks within each batch is uncertain, which is revealed as the retrieval proceeds.

A6: For each batch, the full arrival order of trucks from this batch is revealed at once after all containers in its prior batch have been retrieved.

It is worth mentioning that A6 follows the assumption of Galle et al. (2018b), which is based on the phenomenon of truck congestion at gates and yards in busy terminals. The considerable number of trucks in the queue enables the terminal operator to have information about the full arrival order of trucks in a batch before the retrieval of this batch begins.

3.1.2 Probabilistic model of truck arrivals

The specific probability distribution of the arrival order of trucks within each batch is hard to predict. The existing studies assume a uniform distribution. In the practical situation, trucks (customers) have their preferred arrival times and may have preferences for either the earlier segment or the latter segment of the booked time windows, which leads to unequal probabilities of arriving in each segment.

We propose a more general probabilistic model of truck arrivals, which can capture customer preference-based arrivals. We divide each appointment time window into two sub-time windows with identical time length. More generally, our proposed approach can be applied to the case where each time window is divided into multiple (more than two) sub-time windows. For the sake of brevity and noticing that the current TAS usually sets 30 min or 60 min for each appointment time window, we only focus on the case of two sub-time windows in this paper. The following assumption is made in the probabilistic model.

A7: 1) Within each batch, the probability of a truck arriving at which sub time window is dependent on customer preference, and 2) within each sub time window, the truck arrival order is drawn from a uniform random permutation.

This enables us to list all potential scenarios of the assignments of a batch of trucks to two sub-time windows with associated probabilities calculated by the customer preference. The calculation is presented in the next sub-section along with the introduction of containers' attributes.

3.1.3 Containers' attributes

We introduce containers' attributes to help describe the problem. The following notations are adopted throughout the paper. Let B_k denote the set of containers in batch k and C_k denote the number of containers in batch k, $k \in \{1,...,K\}$. By definition $\sum_{k=1}^{K} C_k = C$. Each container has three attributes:

(1) The first attribute, denoted by l_i, $i \in \{1,...,C\}$, is the ***priority label*** that represents i) the arrival precedence relationship among the trucks and ii) the container retrieval sequence. This label changes during the container retrieval process. Initially, containers in batch k are labeled by L_k that represents the arrival precedence among batches of trucks, which we call ***batch priority*** (see Fig. 1(a)). Let $L_k = 1 + \sum_{j=1}^{k-1} C_j$, such that given L_k, there is a unique $k \in \{1,...,K\}$. Then, once the full arrival order of trucks in batch k is revealed, we get the ***sub-batch priority*** for batch k, which represents the arrival precedence among sub-batches of trucks. A ***sub-batch*** of trucks is the set of trucks that have arrived in the same sub-time window. For a container in batch k, $k \in \{1,...,K\}$, if its corresponding truck is revealed to arrive in the second sub-time window, its label changes to $L_k + 1$; otherwise, its label does not change. Once the retrieval sequence of a container in batch k is determined, its label changes to the exact retrieval sequence that is within $[L_k, L_k + C_k - 1]$.

(2) The second attribute, denoted by u_i, $i \in \{1,...,C\}$, is a ***unique ID,*** which is used for identifying individual containers (trucks) (see Fig. 1(b)).

(3) The third attribute, denoted by $p_i \in [0,1]$, $i \in \{1,...,C\}$, represents the ***customer preference*** of container u_i (see Fig. 1(c)). We define that truck u_i arrives at the first sub-time window with p_i and at the second sub-time window with $1 - p_i$. For $k \in \{1,...,K\}$, let ζ_k refer to the possible scenario of sub-batches of trucks in batch k and $p(\zeta_k)$ refer

to its probability. ζ_k^1 and ζ_k^2 represent two mutually exclusive and collectively exhaustive random sets, the random variables in which take values in $u_i \in B_k$, such that $\{u_i \in \zeta_k^1\}$ is the event that truck u_i arrives in the first sub-batch and $\{u_i \in \zeta_k^2\}$ is the event that it arrives at the second sub-batch. Then, by definition, we have $\mathbb{P}[u_i \in \zeta_k^1] = p_i$ and $\mathbb{P}[u_i \in \zeta_k^2] = 1 - p_i$, $u_i \in B_k$. There are a total of 2^{C_k} possible scenarios of ζ_k for batch *k*, and a total of $\prod_{k=1}^{K} 2^{C_k}$ scenarios for all the batches.

A simple example is given in Fig. 1 to illustrate the containers' attributes and the calculation of the probability of ζ_k. There are seven containers in the initial bay that consists of three stacks and three tiers. Fig. 1(a) displays the priority labels represented by batch priority (1, 3, 5). Fig. 1(b) gives the container/truck ID ($u_1 \sim u_7$). Fig. 1(c) presents the customer preference (0.0~1.0). Fig. 1(d) displays the revealed sub-batch priority of the first batch in bold. Given the information in Fig. 1(a)-(c), we have the initial bay configuration. For example, container u_1 is located in the third tier of stack three; truck u_1 is in the first batch, which will arrive in the first sub-time window with a probability of 0.6 and the second sub-time window with a probability 0.4. Let us consider ζ_1. There are totally four scenarios of ζ_1, which are respectively $\{\zeta_1^1 = \{u_1\}, \zeta_1^2 = \{u_4\}\}$, $\{\zeta_1^1 = \{u_1, u_4\}, \zeta_1^2 = \varnothing\}$, $\{\zeta_1^1 = \{u_4\}, \zeta_1^2 = \{u_1\}\}$, and $\{\zeta_1^1 = \varnothing, \zeta_1^2 = \{u_1, u_4\}\}$. Their probabilities are computed as: $p(\{\zeta_1^1 = \{u_1\}, \zeta_1^2 = \{u_4\}\}) = 0.6 \times (1 - 0.8) = 0.12$; $p(\{\zeta_1^1 = \{u_1, u_4\}, \zeta_1^2 = \varnothing\}) = 0.6 \times 0.8 = 0.48$; $p(\{\zeta_1^1 = \{u_4\}, \zeta_1^2 = \{u_1\}\}) = (1 - 0.6) \times 0.8 = 0.32$; $p(\{\zeta_1^1 = \varnothing, \zeta_1^2 = \{u_1, u_4\}\}) = (1 - 0.6) \times (1 - 0.8) = 0.08$. If truck u_1 has arrived at the terminal in the first sub-time window and truck u_4 has arrived in the second sub-time window as shown in Fig. 1(d), ζ_1 is revealed to be $\{\zeta_1^1 = \{u_1\}, \zeta_1^2 = \{u_4\}\}$.

Tier				Stack
3			1	
2	3	5	1	
1	5	5	3	
	1	2	3	

(a) Batch priority

Tier				Stack
3			u_1	
2	u_2	u_3	u_4	
1	u_5	u_6	u_7	
	1	2	3	

(b) Container (truck) ID

Tier				Stack
3			0.6	
2	0.7	0.5	0.8	
1	0.3	0.1	0.4	
	1	2	3	

(c) Customer preference

Tier				Stack
3			**1**	
2	3	5	**2**	
1	5	5	3	
	1	2	3	

(d) Sub-batch priority

Fig. 1 An illustration of containers' attributes

3.1.4 Service policy

As an alternative to the FCFS policy, we propose a flexible service policy that allows Out-Of-Order retrievals for containers in the same Sub-batch, which is called the SOOO policy. Under this policy, a former sub-batch of trucks is surely served before a latter sub-batch of trucks, and the service sequence for trucks in the same sub-batch is to be determined by the terminal operators. As the root cause of relocation is the mismatch between containers' stacking positions and their retrieval sequences, relocations can be reduced by optimizing the retrieval sequence. Besides, as relocation operations increase retrieval times, out-of-order retrievals can also create opportunities for reducing the truck waiting time in the retrieval process. Similar to Galle et al. (2018b), we make the following assumption on the retrieval service begin time of a batch.

A8: The retrieval service of a batch begins at the end of the appointed time window associated with the batch.

A8 can be justified from the following two aspects. Firstly, A8 represents the practical situation of crowded terminals in which trucks often queue up at gates and yards after their arrivals and wait to be served (see Pham et al., 2011; Chen et al., 2013a,b). On one hand, several activities, e.g., security check, permission check, etc. (see Huynh and Zumerchik, 2010) need to be performed at the entry gates before the truck can proceed to the yard. On the other hand, for container terminals having a high level of congestion in the yard, internal waiting queues are also formed and trucks have to wait

to be served (Talley and Ng, 2016; Li et al., 2019). Secondly, it is observed that the average truck turnaround time could be much longer than the length of the appointment time window. For example, in Los Angeles-Long Beach, the average truck turn time at the 12 container terminals for the last two years was above 67 minutes and the maximum value was nearly 100 minutes (Mongelluzzo, 2020). For some terminals, a truck appointment system with 30-minutes time windows has been implemented (e.g., Fenix Marine Services container (fenixmarineservices.com), Middle Harbor (Mongelluzzo, 2016)). Given above, it is reasonable to assume that the service of a batch begins at the end of the appointed time window associated with the batch.

3.2 Problem formulation

We propose two mathematical models for the SCRP-FS. First, a Sooo model is developed with the objective of minimizing the expected total number of relocations to retrieval all containers from a given bay. The Sooo model is important to the terminal operators in reducing relocations; however, it does not consider the truck waiting times. Second, we develop a Sooo extension model to fully take advantage of the flexible service policy. The Sooo extension model has two lexicographically ordered objectives: the primary objective is to minimize the expected total number of relocations, and the second objective is to minimize the total truck waiting times in each batch. To a large extend, our study on the Sooo model is a starting point to develop the Sooo extension model, which is one of the main contributions of this paper. Still, the results of the Sooo are of a certain interest in their own right, and the developed algorithm serves as building blocks for the exact algorithm for the Sooo extension model.

3.2.1 Sooo model

The SCRP is a multi-stage sequential decision-making problem with dynamic information revealing. The stochastic dynamic programming (SDP) method is appropriate to deal with such problems (Bakker et al., 2020). The CRP related problems have been tackled using (stochastic) dynamic programming method, e.g. the deterministic CRP (Kim and Hong, 2006), the SCRP (Ku and Arthanari, 2016a), and the export container stacking problem (Kim et al., 2000; Zhang et al., 2010). In this paper, we formulate the SCRP-FS into an SDP model. The emphasis in SDP is typically in identifying the system states and the actions (variables) at each state (Birge and Louveaux, 2011). In the following, we first define the stage, state, and action for the Sooo model.

Stage: the sequence number of the batch to be retrieved. The example in Fig. 1 is considered as stage 1 since the *1*st batch of containers is to be retrieved.

State: the state of each stage is the state of the bay that consists of the stacking positions of the remaining containers and their attributes. The input state of the *k*th stage is the state of the bay after the (*k*-1)th batch has been retrieved and before the scenario of the sub-batches of the *k*th batch is revealed. For example, Fig. 1(a)-(c) constitute the input state of stage 1.

Action: a feasible action is defined as a sequence of moves to retrieve a batch of containers. Different from the conventional SCRP, the actions in the SCRP-FS consists of two types of actions: (i) the retrieval sequences of the containers (i.e., the service sequence of trucks) in each of the two sub-batches, called *sequencing*, and (ii) the storage positions of the relocated containers, called *relocating*.

With these definitions, optimal actions are taken under uncertainty stage by stage. The uncertainty in the model refers to the scenarios of the sub-batches of each batch (at each stage). At the beginning of a stage, firstly, the scenario of this stage is revealed, and then the optimal actions to retrieve the batch of containers at this stage are sought accordingly considering all the potential scenarios of future stages. The objective is to minimize the expected total number of relocations to retrieve all the containers. Mathematically, the Sooo model can be formulated in a similar way in which it is done in Ku and Arthanari (2016a). The notations used in the model are defined as follows.

K: the total number of batches in the initial bay, which is also the total number of stages.

k: the stage number (the kth batch of containers to be retrieved), $k \in \{1,...,K\}$.

ζ_k : The scenario of the sub-batches of stage k, $k \in \{1,...,K\}$ (a random variable).

S_k : the input state of stage k, that is, the state of the bay after the (k-1)th batch has been retrieved and before ζ_k is

revealed, $k \in \{1,...,K\}$.

$p(\zeta_k)$: The probability of ζ_k. This is calculated by the probabilistic model of truck arrivals introduced in section 3.1.2.

$\mathbf{a}_k(S_k,\zeta_k)$: The actions (a decision variable) taken for retrieving the *k*th batches of containers given S_k and ζ_k. $\mathbf{a}_k(S_k,\zeta_k)=\{\mathbf{a}_k^S(S_k,\zeta_k),\mathbf{a}_k^R(S_k,\zeta_k)\}$, wherein $\mathbf{a}_k^S(S_k,\zeta_k)$ is the retrieval sequence decision of the containers in each of the two sub-batches at stage *k* given S_k and ζ_k, and $\mathbf{a}_k^R(S_k,\zeta_k)$ is the relocation position decision that respects $\mathbf{a}_k^S(S_k,\zeta_k)$. For notational convenience, the dependence on (S_k,ζ_k) is suppressed from $\mathbf{a}_k(S_k,\zeta_k)$, and we use $\mathbf{a}_k$ instead.

$r_k(\mathbf{a}_k \mid S_k,\zeta_k)$: The number of relocations that are required during action $\mathbf{a}_k$ on the bay of state S_k given ζ_k.

$t_k(S_k,\zeta_k,\mathbf{a}_k)$: The state transition function that maps S_k, ζ_k, and $\mathbf{a}_k$ into the next state S_{k+1}. By $t_k(S_k,\zeta_k,\mathbf{a}_k)$, the *k*th batch of containers revealed by ζ_k are retrieved according to $\mathbf{a}_k$ from state S_k, after which S_{k+1} is obtained.

$f_k(S_k)$: The expected minimum total number of relocations to retrieve the remaining *K*-*k*+1 batches of containers from the state S_k.

The Sooo model is formulated as a recursive equation as follows:

$$f_1(S_1)=E\left[\min_{\mathbf{a}_1}\{r_1(\mathbf{a}_1 \mid S_1,\zeta_1)+f_2(S_2)\}\right]=\sum_{\zeta_1}p(\zeta_1)\min_{\mathbf{a}_1}\left[r_1(\mathbf{a}_1 \mid S_1,\zeta_1)+f_2(S_2)\right],$$

where $S_2=t_1(S_1,\zeta_1,\mathbf{a}_1)$ (1)

Generally, $f_k(S_k)=\sum_{\zeta_k}p(\zeta_k)\min_{\mathbf{a}_k}\left[r_k(\mathbf{a}_k \mid S_k,\zeta_k)+f_{k+1}(S_{k+1})\right],\ k\in\{1,...,K\}$,

where $S_{k+1}=t_k(S_k,\zeta_k,\mathbf{a}_k)$, for $k\in\{1,...,K\}$, and $f_{K+1}(S_{K+1})=0$ (2)

The recursive function of equation (2) indicates that optimal decisions can be obtained by optimizing the recursive function in a backward manner stage by stage.

3.2.2 Sooo extension model

The Sooo extension model considers two lexicographically ordered objectives. The primary objective is to minimize the expected total number of relocations and the secondary objective is to minimize the total truck waiting times of each batch sequentially. The use of the secondary objective is justified from the following three perspectives.

Firstly, our motivation for considering the metric of truck waiting times stems from its importance not only to the container terminals but also to the container transport supply chain. The truck waiting time is a key performance indicator that measures the efficiency of storage area at a container terminal (Stahlbock and Voß, 2008; Carlo et al., 2014; Gharehgozli et al., 2016) and is one of the main reasons causing delays in handling external trucks and leading to low quality of customer service (Borjian et al. 2013). A reduction in the truck waiting time would improve the terminals' competitiveness and act as an incentive to encourage external truckers' cooperation, which is essential to achieve a smooth implementation of the flexible service policy.

Secondly, longer truck waiting time in the yard for service leads to higher truck turn time and more emissions (Huynh et al., 2004). Terminal operators have been under enormous pressure from different parties requiring to reduce the truck turn time. For example, from the legislative perspective, the California Assembly Bill AB 2650 became active in 2003 in the US requiring port terminals to lower port-related truck congestion and vehicle emissions. Under this law, external trucks were a major target of regulatory efforts (Giuliano and O'Brien, 2007). Besides, from the economic perspective, some port authorities (e.g., Port Botany; Port Metro Vancouver) have implemented a penalty system that imposes fees on terminals that exceed a specified threshold of truck turn time. In addition, from the perspective of vertical cooperation in the container transport chain, truckers, as an import stakeholder of the hinterland transport, have stated that they won't accept truck appointments until terminals can shorten turn times and end long queues (Bonney, 2015). Reducing the truck waiting time in the container retrieval process helps to alleviate yard congestion and thus reduce the truck turn time.

Thirdly, the importance of the truck waiting time metric in the CRP has also been confirmed by the increasing attention it has received in the literature. For instance, López-Plata et al. (2017) minimize the total waiting times of the containers

that have expected retrieval times in the deterministic CRP. Borjian et al. (2015b) and Borjian et al. (2013) minimize the weighted sum of total relocations and delays in the deterministic CRP and uncertain CRP respectively. Our study is the first that considers two lexicographically ordered objectives when taking both the number of relocations and the trucks waiting time into consideration in the uncertain CRP. The reasons to sequence these two objectives (i.e. the number of relocations as the primary objective and the truck waiting time as the secondary objective) are: (i) the number of relocations has a more direct impact on the terminal; (ii) this treatment will appropriately evaluate the effect of the flexible policy on reducing the expected number of relocations compared to the conventional SCRP; (iii) because there may exist multiple optimal solutions to minimizing the expected number of relocations in the SCRP-FS, introducing the secondary objective can further optimize the second objective without sacrificing the primary objective.

Fig.2 illustrates the idea behind the Sooo extension model. Let us consider the solutions for the first batch, in which truck u_1 and u_5 have been revealed to be in the first sub-batch and u_7 in the second sub-batch. With regard to the primary objective, there are two optimal solutions to the retrieval sequence for u_1, u_5, and u_7. Solution one is $u_5 \rightarrow u_1 \rightarrow u_7$, and solution two is $u_1 \rightarrow u_5 \rightarrow u_7$. The two solutions contribute the same number of relocations to the expected total number of relocations since no matter which retrieval sequence is used the blocking container u_2 is relocated to stack two. The Sooo extension model wants to choose one that is optimal with respect to the secondary objective, i.e., minimizing the total waiting times of the trucks in the first batch/stage. By A8, both truck u_1 and u_5 are ready to be served when the service of this batch begins. If u_5 is retrieved before u_1, truck u_1 suffers waiting due to the relocation of u_2, which could have been avoided if using the alternative solution. Therefore, the optimal solution of the Sooo extension model is $u_1 \rightarrow u_5 \rightarrow u_7$.

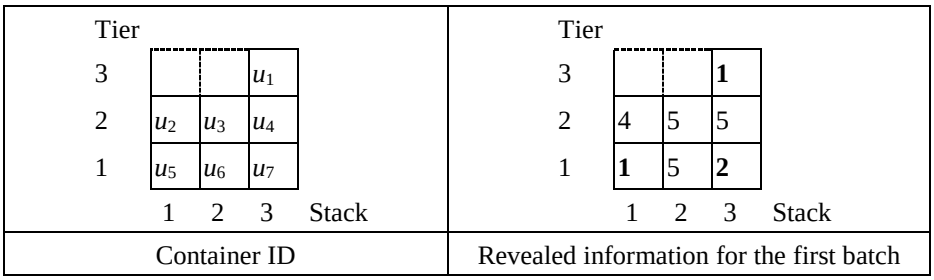


Fig. 2 An example of illustrating the primary objective and the secondary objective

Objective function

Here, we develop the secondary objective function. Before getting retrieval service, truck waiting can occur at any point from arriving outside the in-gate to arriving at the designated yard stack until exiting the out-gate. As our main focus is on the container retrieval process, only the yard-to-retrieval waiting time is of our interest. In particular, we are more concerned with how much waiting times in the container retrieval service process can be saved as a result of the flexible service policy as opposed to the FCFS policy. By the assumption A8, the retrieval service for a truck cannot commence before the end of its appointed time window. Therefore, the waiting time of a truck before the end of its appointed time window is independent of our decision variables. We hence define the truck waiting time as the elapsed time between the end of its appointed time window and its actual retrieval time. To avoid confusion, we use the term "relevant truck waiting time" to represent the truck waiting time considered in this paper. It is worthwhile to note that the relevant truck waiting time under A8 has its practical interpretation. In practice, a truck would have an expected time to retrieve its container, and the difference between that and the actual retrieval time is a common measure of service quality in general terms (López-Plata et al., 2017). For a truck that has booked an arrival time window through the TAS, the end of its appointed time window can be regarded as its expected retrieval time, and the relevant truck waiting time can be used as a measure of service quality for the container retrieval process.

To measure the relevant truck waiting times, we need some time-based notations. Let e_k denote the end of the appointed time window of the containers in batch k. Let d_k denote the completion time of retrieving the last container in batch k, and y_k denote the service starting time of batch k. The secondary objective is optimized for each stage

separately in a sequential way rather than considering global optimization. This means that when optimizing the secondary objective of stage k, the service for stage k-1 has been completed, that is, S_k is given. This enables us to treat the service completion time of the (k-1)th stage (d_{k-1}) as a constant when optimizing stage k. Given e_k and d_{k-1}, we have y_k by

$$y_k = \max\{e_k, d_{k-1}\},\ k \in \{2,...,K\};\ y_1 = e_1. \tag{3}$$

By equation (3), if the service completion time of batch k-1 is later than the end of the appointed time window of batch k, the service starting time of batch k is d_{k-1}. Otherwise, the service starting time of batch k coincides with the end of its appointed time window, that is, e_k.

Under the given decisions $\mathbf{a}_k(S_k, \zeta_k) = \{\mathbf{a}_k^S(S_k, \zeta_k), \mathbf{a}_k^R(S_k, \zeta_k)\}$, we now derive the explicit expressions of d_k and the relevant waiting time of truck i in batch k ($i \in B_k$) under ζ_k, which is denoted by $w_{k,i}^{\zeta_k}$. The following notations are introduced to extract the relevant information implied in the decision variables.

$o_i^{\zeta_k}$: the service order of truck i, $i \in B_k$, under ζ_k. Note that $o_i^{\zeta_k}$ is implied in the service sequence decision $\mathbf{a}_k^S(S_k, \zeta_k)$.

$r_i^{\zeta_k}$: the number of relocation moves needed when serving truck i, $i \in B_k$, under ζ_k. Note that $r_i^{\zeta_k}$ is implied in the relocation decision $\mathbf{a}_k^R(S_k, \zeta_k)$.

t^{ret} : the handling time per retrieval move.

t^{rel} : the handling time per relocation move.

By A8, all the trucks in a batch have already waited at the yard stack when the service of this batch begins, and thus there is no idle time between the services of any two trucks in the batch. Therefore, d_k is calculated by

$$d_k = y_k + \sum_{i \in B_k} (t^{rel} \cdot r_i^{\zeta_k} + t^{ret}) \tag{4}$$

$w_{k,i}^{\zeta_k}$ is calculated by

$$w_{k,i}^{\zeta_k} = (y_k - e_k) + \sum_{j \in B_k, o_j^{\zeta_k} < o_i^{\zeta_k}} (t^{rel} \cdot r_j^{\zeta_k} + t^{ret}) + t^{rel} \cdot r_i^{\zeta_k}, \tag{5}$$

The first term on the right side in equation (5) is the waiting time between the end of the appointed time window and the start of the service of batch k, the second term is the total handling time of the trucks in the batch that are served before truck i, and the last term is the relocation time for retrieving the container requested by truck i. Equation (5) is illustrated in Fig. 3 using the instance in Fig. 2, where the first batch of trucks is served in the sequence: $u_1 \rightarrow u_5 \rightarrow u_7$. By equation (5), the waiting time of truck u_1, u_5, and u_7, is respectively $y_k - e_k$, $y_k - e_k + t^{ret} + t^{rel}$, and $y_k - e_k + 2t^{ret} + 2t^{rel}$. It is observed that the handling time of u_1 contributes t^{ret} to the waiting time of both u_5 and u_7 and the handling time of u_5 contributes $t^{ret} + t^{rel}$ to the waiting time of u_7.

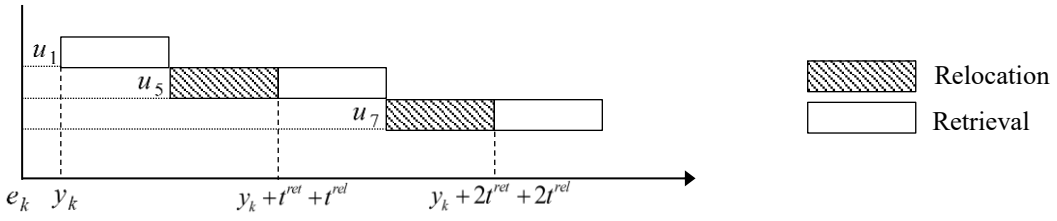


Fig. 3 Illustration of trucks waiting time

Let $W_k^{\zeta_k}$ denote the total waiting times of all the trucks in batch k under ζ_k with the given decisions $\mathbf{a}_k(S_k, \zeta_k)$. Then we have

$$W_k^{\zeta_k} = \sum_{i\in B_k} w_{k,i}^{\zeta_k} = \sum_{i\in B_k}\left((y_k - e_k) + \sum_{j\in B_k, o_j^{\zeta_k} < o_i^{\zeta_k}} (t^{rel}\cdot r_j^{\zeta_k} + t^{ret}) + t^{rel}\cdot r_i^{\zeta_k}\right),$$
$$= (y_k - e_k)\cdot C_k + \sum_{i\in B_k}(t^{rel}\cdot r_i^{\zeta_k} + t^{ret})\cdot\left(C_k - o_i^{\zeta_k}\right) + \sum_{i\in B_k} t^{rel}\cdot r_i^{\zeta_k} \quad (6)$$

where the first term on the right hand represents the total waiting time of all trucks in the batch before the service of this batch commences, the second term represents the sum of the handling times of the predecessors of each truck in the batch, and the third term is the total relocation times for retrieving all the containers in the batch. Because the service time of truck *i* adds to the waiting times of all its successors in the batch, $(t^{rel}\cdot r_i^{\zeta_k} + t^{ret})$ is weighted by $(C_k - o_i^{\zeta_k})$. In other words, a part of the waiting time of a truck is a cumulative service time of all its predecessors in the batch (see Fig. 3).

Let $\gamma_k^{(1)}(S_k)$ denote the primary objective for stage *k*, which is the expected minimum total number of relocations to retrieve the remaining *K*-*k*+1 batches of containers from the state S_k, and $\gamma_k^{(2)}(S_k)$ denote the secondary objective for stage *k*, which is the expected minimum total waiting times for the trucks in batch *k* with the state S_k. Given S_k, the Sooo extension model aims to find the solution that minimizes the secondary objective $\gamma_k^{(2)}(S_k)$ among the set of solutions that minimize the primary objective $\gamma_k^{(1)}(S_k)$. Then, the Sooo extension model for stage *k* with the state S_k, $k \in \{1,...,K\}$, is formulated as follows:

$$\gamma_k^{(1)}(S_k) = f_k(S_k), \text{ which is defined in (2)}$$
$$\gamma_k^{(2)}(S_k) = \sum_{\zeta_k} p(\zeta_k) \min_{\mathbf{a}_k} W_k^{\zeta_k} \quad (7)$$

Derivation of optimality

Observation 1. The optimal solution of the Sooo extension model at stage *k* from the state S_k under ζ_k is the one that minimizes $\sum_{i\in B_k} r_i^{\zeta_k}\cdot\left(C_k - o_i^{\zeta_k} + 1\right)$ among the set of optimal solutions $\mathbf{a}_k(S_k, \zeta_k)$ with regard to the primary objective.

Proof. At each stage *k*, $d_{k\text{-}1}$ is known and thus y_k can be calculated by equation (3). Besides, e_k, C_k, t^{ret} and t^{rel} are constant. Therefore, $\min_{\mathbf{a}_k} W_k^{\zeta_k}$ in equation (7) is equivalent to

$$\min_{\mathbf{a}_k} \sum_{i\in B_k} r_i^{\zeta_k}\cdot\left(C_k - o_i^{\zeta_k} + 1\right), \quad (8)$$

where $r_i^{\zeta_k}\cdot\left(C_k - o_i^{\zeta_k} + 1\right)$ represents the contribution of the number of relocations for retrieving container *i* on the waiting time of truck *i* itself and on the waiting times of all its successors in the batch. □

Observation 1 indicates that at each stage, the secondary objective is jointly determined by the number of relocations for each retrieval in the batch and its retrieval sequence.

4. Exact solution algorithms based on decision tree

By applying the recursive equation of the Sooo model, the optimal solutions can be obtained backward from stage *K* to stage 1. This procedure is usually executed by a tree search-based algorithm in a state-space constructed by a decision tree. The state-of-the-art tree search-based algorithm for the SCRP in which information is revealed on a batch basis is the Pruning-Best-First-Search (PBFS) algorithm proposed by Galle et al. (2018b). To solve the Sooo model, we develop an Adapted Pruning-Best-First-Search (APBFS) algorithm by extending the PBFS algorithm; we then extend the APBFS algorithm to solve the Sooo extension model. The PBFS is developed for the FCFS rule and does not consider customer

preference, which does not allow for a straightforward adaption. The main differences between our algorithms from existing ones are explained below. Firstly, our decision tree is more general, which can support two types of decision-making under flexible service policies and incorporate customer preference. In particular, our decision tree is composed of three classes of nodes and each node is represented by a dual-matrix configuration, which differs from the PBFS decision tree that is composed of two classes of nodes each represented by a single-matrix configuration. Secondly, the new structure of the decision tree necessitates several major adaptions in the search: first, a new decision layer is considered during searching and back-tracking; second, another two techniques used to decrease the search space are modified: the abstract technique and the lowest level to stop branching. Thirdly, a new lower bound for the expected number of relocations for the SCRP-FS is proposed to prune unnecessary nodes. We consider the new lower bound as one of the major contributions of this paper since this is the first lower bound for the SCRP-FS in the literature. Fourthly, our extended APBFS expands the PBFS by having the capability of minimizing the total truck waiting times of each batch, which has been neglected in the relevant literature.

In sub-sections 4.1-4.3, we first introduce the elements of the proposed algorithms with a focus on the differences between the PBFS and the APBFS. Then, in sub-section 4.4 and 4.5, we present the APBFS for the Sooo model and the extended APBFS for the Sooo extension model respectively. The contribution of the extended APBFS is introduced in sub-section 4.5.

4.1 Constructing a decision tree

In a typical decision tree for the SCRP, the root node represents the initial configuration and the leaf nodes represent the empty configuration. Between the root node and leaf nodes, there are two types of intermediate nodes: chance nodes and decision nodes, which alternate in some way to form the tree. A chance node is to model the stochasticity of trucks' arrival while a decision node is to model possible actions. In the PBFS, each node is represented by a single matrix that corresponds to the truck arrival orders.

In the (extended) APBFS algorithm, each node is represented by a ***dual-matrix configuration*** that is composed of a priority matrix and a preference matrix (see Fig. 4). The ***priority matrix*** represents the priority labels of containers and the ***preference matrix*** represents the probability of trucks arriving in the first sub-time window (i.e. the customer preference). The structure of the decision tree consists of ***three classes of nodes***, which are ***chance nodes***, ***SD nodes*** (sequencing decision nodes) and ***RD nodes*** (relocating decision nodes). The SD nodes create a new decision layer between the chance nodes and the RD nodes to sequence trucks. In the following, we define these nodes with the introduction of relevant notations.

A ***chance node*** corresponds to S_k, $k \in \{1,...,K\}$, in which the scenario of sub-batches of batch *k* (i.e., ζ_k) is to be revealed (see e.g., node 0 in Fig. 4). From a chance node, descendant nodes are created by ζ_k, denoted by S_k' (see e.g., node 1). Let $\xrightarrow{\zeta_k}$ represent the revelation of ζ_k, we have $S_k \xrightarrow{\zeta_k} S_k'$, $k \in \{1,...,K\}$.

After the revelation of the random variable, actions are taken to retrieve the containers in batch *k* from S_k'. A ***SD node*** corresponds to S_k', in which the retrieval sequence for the *k*th batch, denoted by D_k^S, is to be determined. From a SD node, descendant nodes, denoted by X_{L_k} (e.g., node 5, node 6), are created by applying D_k^S. Recall that as defined in section 3.1.3, L_k represents the batch priority such that each batch *k* corresponds to an unique L_k. Let $\xrightarrow{D_k^S}$ represent the application of D_k^S, we have $S_k' \xrightarrow{D_k^S} X_{L_k}$. In a ***RD node***, the target container (the container with the smallest label) is to be retrieved and the relocation decisions to retrieve the container is to be determined. For batch *k*, starting from the SD node corresponding to S_k', C_k levels of RD nodes are created sequentially, denoted by X_i, $i \in \{L_k,...,L_k + C_k - 1\}$ (e.g., node 5, 7). Let D_i^R denote a sequence of moves (relocation moves and retrieval move) to retrieve the *i*th container, and X_{i+1} denote the configuration after applying action D_i^R to X_i and before applying action D_{i+1}^R, $i \in \{L_k,...,L_k + C_k - 1\}$. Let $\xrightarrow{D_i^R}$ represent the application of D_i^R, we have $X_i \xrightarrow{D_i^R} X_{i+1}$,

$i \in \{L_k, ..., L_k + C_k - 2\}$.

After the retrieval of the last container in the kth batch whose retrieval sequence is $L_k + C_k - 1$, $X_{L_k + C_k - 1}$ transits to the next chance node corresponding to S_{k+1} (e.g., node 10), which is represented by $X_{L_k + C_k - 1} \xrightarrow{D^R_{L_k + C_k - 1}} S_{k+1}$.

To summarize, for $k \in \{1, ..., K\}$, the state transitions from S_k to S_{k+1} in the tree search are modeled by:

$$\text{if } C_k > 1, \begin{cases} S_k \xrightarrow{\zeta_k} S_k' \\ S_k' \xrightarrow{D^S_k} X_{L_k} \\ X_i \xrightarrow{D^R_i} X_{i+1}, \forall i \in \{L_k, ..., L_k + C_k - 2\} \\ X_{L_k + C_k - 1} \xrightarrow{D^R_{L_k + C_k - 1}} S_{k+1} \end{cases}$$

$$\text{if } C_k = 1, \; S_k \xrightarrow{D^R_{L_k}} S_{k+1} \tag{9}$$

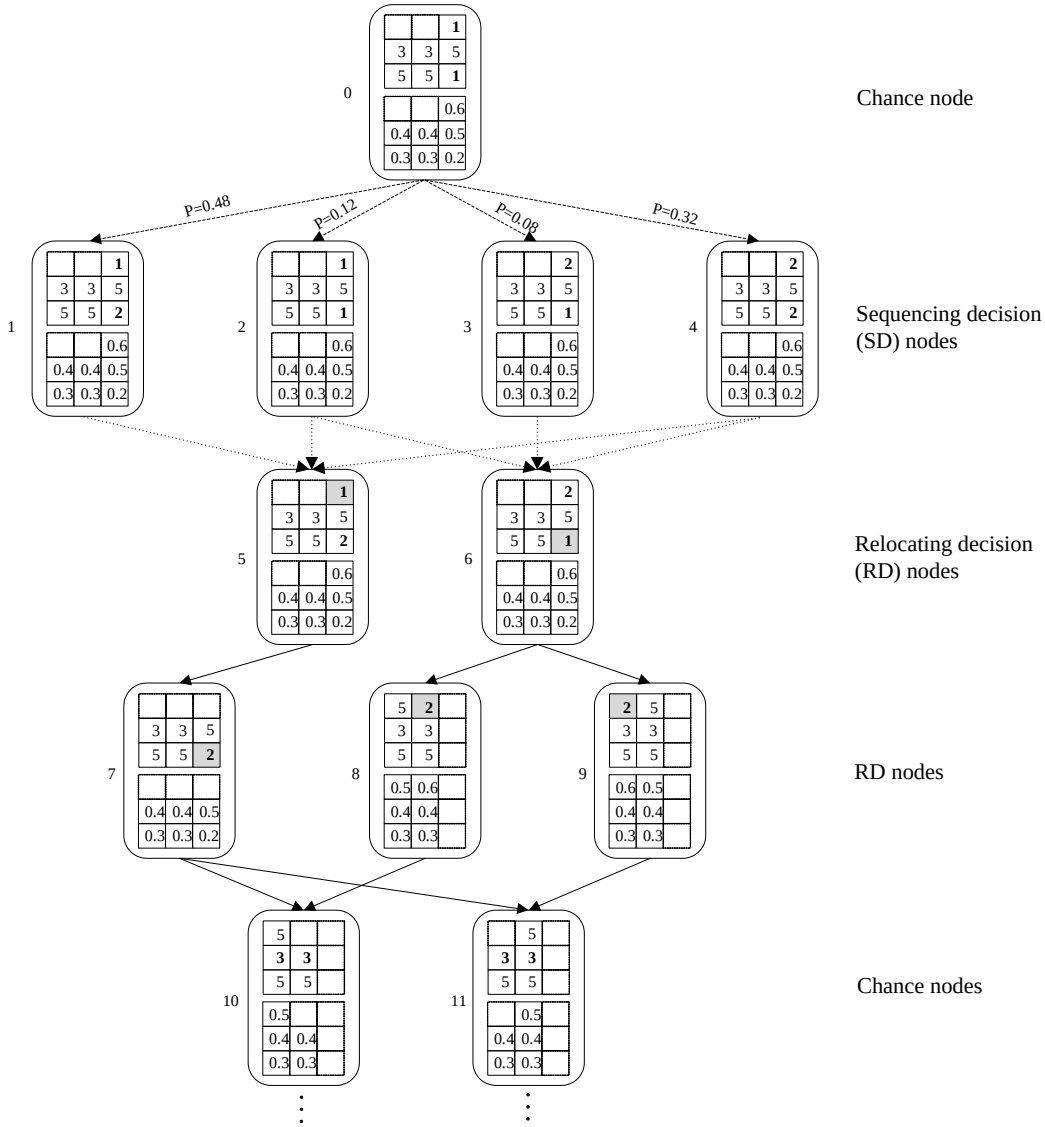


Fig. 4 A sample decision tree

Dashed-lines: the revelation of the scenario of sub-batches; Dotted-lines: applying the sequencing action on retrieval order; Solid-lines: applying the relocating action and retrieving the containers; Containers in bold font: target batch of containers; Containers in the shaded slot: target container to be retrieved.

4.2 Back-tracking in the decision tree

Given a full decision tree, the optimal objective value is calculated by back-tracking. In this section, we only focus on the primary objective. Let n be a node in the decision tree. Each node has a cost-to-go function, denoted by $f(n)$, which represents the expected cost of the cheapest path from node n to the leaf node. Let n=0 denote the root node, and then the objective function is denoted by $f(0)$. The basic idea of back-tracking is to compute the cost-to-go function $f(n)$ for each node n recursively from the bottom up of the tree with the ultimate goal to obtain $f(0)$. To calculate $f(n)$, we need the immediate cost function, denoted by $r(n)$, which represents the cost incurred by the action taken to transit node n to its offspring. In the SCRP-FS, $f(n)$ is defined for all three types of nodes, which represents the minimum expected number of relocations required to retrieve all remaining containers from node n. $r(n)$is only defined for RD nodes, which represents the number of relocations required in order to retrieve the target container in node n. The following notations are used to calculate $f(n)$.

λ_n : the number of remaining containers in node n, which is defined as the level of n. If n is a chance node or a SD node, there exists a unique $k \in \{1,...,K\}$ such that $L_k = C - \lambda_n + 1$.

Ω_n : the set of offspring of a chance node n. Each node in Ω_n is a SD node that corresponds to a realization of the random variable ζ_k , and thus $|\Omega_n| = 2^{C_k}$.

Φ_n : the set of offspring of a SD node n. Each node in Φ_n corresponds to a feasible retrieval order for the containers in batch k, wherein $L_k = C - \lambda_n + 1$, and thus $|\Phi_n| = |\zeta_k^1|!|\zeta_k^2|!$.

Δ_n : the set of offspring of a RD node n. Δ_n is constructed greedily by considering all feasible combinations of the relocation positions of the $r(n)$ blocking containers in node n. The maximum value of $|\Delta_n|$ is given by $(S-1)^{r(n)}$ when the number of empty slots in each candidate stack is no less than $r(n)$, wherein S is the number of stacks.

p_{n_i} : the probability of a SD node $n_i \in \Omega_n$, which is calculated by our probabilistic model of truck arrivals.

Given the above definitions, for each node n, we have:

$$f(n) = \begin{cases} \sum_{n_i \in \Omega_n} p_{n_i} f(n_i), \text{if (i) } n \text{ is a chance node,} \\ \min_{n_i \in \Phi_n} \{ f(n_i) \}, \text{if (ii) } n \text{ is a SD node,} \\ r(n) + \min_{n_i \in \Delta_n} \{ f(n_i) \}, \text{if (iii) } n \text{ is a RD node.} \end{cases} \tag{10}$$

In equation (10), the "if (ii)" condition is a new decision layer to the PBFS algorithm. We use the example in Fig. 4 to illustrate equation (10). Suppose $f(10) = f(11) = 1$ is given. Then the $f(n)$ of other nodes are calculated as follows: $f(7) = r(7) + \min\{f(10), f(11)\} = 1 + 1 = 2$; $f(5) = r(5) + \min\{f(7)\} = 0 + 2 = 2$; $f(1) = \min\{f(5)\} = 2$; $f(8) = r(8) + \min\{f(10)\} = 0 + 1 = 1$; $f(9) = r(9) + \min\{f(11)\} = 0 + 1 = 1$; $f(6) = r(6) + \min\{f(8), f(9)\} = 2 + 1 = 3$; $f(2) = \min\{f(5), f(6)\} = 2$. It confirms that the optimal offspring of node 2 with regard to the primary objective is node 5. $f(3) = \min\{f(6)\} = 3$; $f(4) = \min\{f(5), f(6)\} = 2$. It confirms that the optimal offspring of node 4 with regard to the primary objective is node 5. $f(0) = p_1 * f(1) + p_2 * f(2) + p_3 * f(3) + p_4 * f(4) = 0.48*2 + 0.12*2 + 0.08*3 + 0.32*2 = 2.08$.

4.3 Techniques to decrease the size of the decision tree

For larger problems, considering a full decision tree in the tree search becomes computationally cumbersome due to the exponential growth of the search space with the growing size of the problem. In the PBFS, a combination of four techniques has been proposed to reduce the size of the tree, while ensuring the optimality of the solution. The first one is the BFS (Best-First-Search) exploration strategy based on a valid lower bound, which determines the search direction of the tree. The BFS first explores the nodes with smaller lower bound, because these nodes are the most promising nodes that are most likely to return small $f(.)$. The second technique is pruning with a lower bound. By using the pruning strategy, a node is fathomed if its lower bound is greater than or equal to the best $f(.)$ of the explored nodes. The third

technique is stopping the search at a level λ^* at which $f(.)$ can be obtained using specific techniques without the need for further branching. Finally, the abstracting technique is used to avoid re-generating and re-computing identical nodes.

To use these techniques in the APBFS algorithm, we make the following major adaptions. Firstly, we extend the abstract technique by using dual-matrix configurations. Secondly, we derive a new lower bound for the SCRP-FS. Thirdly, we use $\lambda^* = S$ to stop further branching. The following three sub-sections present these adaptions respectively.

4.3.1 Abstraction technique

The abstraction technique is first studied by Ku and Arthanari (2016b) to reduce the search space of the CRP and then is used by Ku and Arthanari (2016a) and Galle et al. (2018b) for the SCRP. The rationale behind this technique is that some configurations are actually equivalent in terms of their contributions to the objective function, and thus duplicate nodes can be avoided. Generally, each newly generated node is abstracted by using a projection rule, after which we determine whether to keep this node or not by comparing it with the nodes that have been explored in the same level. In the (extended) APBFS algorithm, two nodes are regarded as equivalent only when both their abstract priority configurations and abstract preference configurations are identical. This is different from the PBFS. The projection procedure of the abstraction technique for the (extended) APBFS algorithm is as follows, and an illustration is provided in Appendix A.1. We denote the application of this procedure to node n as $Abstract(n)$.

Step 1: Rank the stacks within the priority configuration according to the heights of stacks in ascending order. Ties are broken by ranking them lexicographically in ascending order according to the priority labels of the containers from top tier to bottom tier.

Step 2: (obtain abstract priority configuration): Re-arrange the stacks within the priority configuration according to their rankings so that lower-ranked stacks are located on the left and higher-ranked ones on the right.

Step 3: (obtain abstract preference configuration): Re-arrange the stacks within the preference configuration in the same order of the rankings of the stacks in the priority configuration.

Remark: we observe that executing the abstract technique could be time-consuming as a newly generated node has to be compared with the configurations of all the existing nodes at the same level. For example, it takes about 100 seconds to implement the abstract technique on a node that needs to be compared with 7722 nodes at the same level. Future research may seek more efficient abstract techniques that allow efficient checking for repeated states, such as using a hash table that compares the hash value of two nodes instead of their exact configurations (Russell and Norvig, 2016).

4.3.2 Lower bound

A new lower bound for the SCRP-FS is proposed to prune unpromising nodes. Here we only consider the lower bound on the blocking containers, which is the expected number of containers that must be relocated at least once in order to retrieve all the containers from a node n, denoted by $lb(n)$. We care about the blocking lower bound for RD nodes and chance nodes, while a lower bound for SD nodes is unnecessary because each SD node needs to be explored to return the cost-to-go function of a chance node. In the SCRP-FS, due to the application of the SOOO policy and the incorporation of customer preference, the probability of a container being blocking is different from the conventional SCRP. From now on throughout the paper, we refer to the method of calculating the blocking lower bound for the SCRP-FS as *LB-FS*. In the following, we explain how to compute $lb(n)$ by *LB-FS*.

Lemma 1. Let n be a node with S stacks and T tiers, each stack containing H_s containers ($0 \le H_s \le T$). Let n_h^s denote the container located at stack s ($s \in \{1,...,S\}$) and tier h ($h \in \{1,...,H_s\}$), $l_{n_h^s}$ denote the priority label of container n_h^s, and $p_{n_h^s}$ denote the customer preference for container n_h^s, then we have:

$$lb(n)=\sum_{\substack{s=1,\\ H_s>1}}^{S}\sum_{h=2}^{H_s}\left(1\left\{l_{n_h^s}>\min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}\right\}\times 1+1\left\{l_{n_h^s}=\min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}\right\}\times\left(1-p_{n_h^s}\right)\times\left(1-\prod_{\substack{i=1\\ l_{n_i^s}=l_{n_h^s}}}^{h-1}\left(1-p_{n_i^s}\right)\right)\right) \tag{11}$$

where $1\{A\}$ is the indicator function of A: $1\{A\}$ = 1 if condition A is true; and 0 otherwise.

Proof. The basic idea of computing $lb(n)$ is to compute the expectation of a single container being blocking and then sum up the expectation for all the containers in node n. Let us fix s and compute the probability that container n_h^s is blocking. Clearly, for $H_s=0$, container n_h^s is not blocking for sure. Now we consider $H_s>1$. Obviously, if $l_{n_h^s}<\min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}$, container n_h^s is not blocking. Then, we consider the following two cases in which container n_h^s may be blocking.

(i) If $l_{n_h^s}>\min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}$, then container n_h^s is surely blocking. In this case, the probability that container n_h^s is blocking is equal to 1. In the example of Fig. 5 (a chance node), container n_5^s meets this condition as $l_{n_5^s}=2>\min_{i\in\{1,\ldots,4\}}\left\{l_{n_i^s}\right\}=1$, and thus container n_5^s contributes one blocking container to $lb(n)$.

n_5^s	2	0.6
n_4^s	1	0.0
n_3^s	2	0.2
n_2^s	2	0.4
n_1^s	2	0.5

Fig. 5 Illustration of a single stack configuration for computing the blocking lower bound

(ii) Otherwise, $l_{n_h^s}=\min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}$, which means there are containers below n_h^s with the same label, then container n_h^s is blocking with probability. This case makes the probability a container being blocking different from that in the Galle et al. (2018b). Recall the SOOO policy: the first sub-batch of trucks is given higher service priority over the second sub-batch of trucks; the trucks in the same sub-batch are given the same service priority. Therefore, n_h^s is surely blocking only in the situation where n_h^s belongs to the second sub-batch and there is at least one container with the same label below n_h^s belonging to the first sub-batch. In this condition, the probability that container n_h^s is blocking is equal to $\left(1-p_{n_h^s}\right)\times\left(1-\prod_{\substack{i=1\\ l_{n_i^s}=l_{n_h^s}}}^{h-1}\left(1-p_{n_i^s}\right)\right)$, where $1-p_{n_i^s}$ is the probability that container n_h^s belongs to the second sub-batch, and $1-\prod_{\substack{i=1\\ l_{n_i^s}=l_{n_h^s}}}^{h-1}\left(1-p_{n_i^s}\right)$ is the probability that at least one container with label $l_{n_h^s}$ below n_h^s belong to the first sub-batch. In Fig. 5, container n_2^s and n_3^s meet this condition. $\mathbb{P}\left[n_2^s \text{ is blocking}\right]$ $=(1-0.4)\times(1-(1-0.5))=0.3$; $\mathbb{P}\left[n_3^s \text{ is blocking}\right]=(1-0.2)\times(1-(1-0.4)\times(1-0.5))=0.56$.

Combining the above two cases, we have,

$$\mathbb{P}\left[n_h^s \text{ is blocking}\right] = \mathbb{1}\left\{l_{n_h^s} > \min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}\right\}\times 1 + \mathbb{1}\left\{l_{n_h^s} = \min_{i\in\{1,\ldots,h-1\}}\left\{l_{n_i^s}\right\}\right\}\times\left(1-p_{n_h^s}\right)\times\left(1-\prod_{\substack{i=1\\ l_{n_i^s}=l_{n_h^s}}}^{h-1}\left(1-p_{n_i^s}\right)\right).$$

Summing the above equation for $h\in\{2,\ldots,H_s\}$ and $s\in\{1,\ldots,S\}$, we have equation (11), which completes the proof. □

By equation (11), the $lb(\cdot)$ of the single stack in Fig.5 is calculated as:

$$\mathbb{P}\left[n_2^s \text{ is blocking}\right] + \mathbb{P}\left[n_3^s \text{ is blocking}\right] + \mathbb{P}\left[n_5^s \text{ is blocking}\right] = 0.3+0.56+1=1.86.$$

4.3.3 Lowest level to stop branching

Another technique to decrease the size of the decision tree is stopping branching at an early level λ^* without the need for traversing to the leaf node. λ^* is regarded as the lowest level of the tree. The PBFS algorithm stops branching at $\lambda^* = \max\{S, C_K\}$ and computes $f(.)$ either using a lower bound or A^* algorithm (an efficient algorithm for the classical CRP). Here we use $\lambda^*=S$ and compute $f(.)$ using *LB-FS*. Noticing that the number of containers in a chance node and in its offspring (a SD node) is equal, it does not make sense to stop the search at a SD node, because we can stop the search at the chance node as soon as λ^* is satisfied. In other words, the lowest level of the tree will be reached at either a chance node or a RD node.

Lemma 2. Let n be a chance node or a RD node with S stacks, T tiers, and S containers, and $\lambda_n = \lambda^* = S$, then $lb(n) = f(n)$.

Proof. Since there are only S containers remaining to be retrieved and there are S stacks, any blocking container can be relocated to an empty stack. As a result, the relocated container will never block other containers again. Therefore, each blocking container at node n will have only one relocation in the optimal solution. In addition, each relocation in the optimal solution is unavoidable according to the definition of $lb(n)$. This implies that the optimal solution $f(n) = lb(n)$, which completes the proof. □

4.4 The APBFS algorithm for the Sooo model

Built upon the elements introduced above, we present the whole framework of the Adapted Pruning-Best-First-Search (APBFS) algorithm for the Sooo model. The following notations are used for describing the algorithm.

Ω_n^{APBFS}: the set of offspring of chance node n that is used to compute $f(n)$, which is the subset of Ω_n.

Φ_n^{APBFS}: the set of offspring of SD node n that is used to compute $f(n)$, which is the subset of Φ_n.

Δ_n^{APBFS}: the set of offspring of RD node n that is used to compute $f(n)$, which is the subset of Δ_n.

Give a configuration n and lower bound *LB-FS*, the steps of the APBFS algorithm to return $f(n)$ is as follows.

Algorithm 1. APBFS algorithm $f(n) = APBFS(n, LB\text{-}FS)$

Step 1. Identify the class of node n. If n is a SD node, go to Step 4. Otherwise, go to Step 2.

Step 2. If the level of n is not greater than S, compute $f(n)$ using $lb(n)$. Otherwise, go to Step 3.

Step 3. If n is a chance node, compute $f(n)$ following Step 3.1~3.3.

Step 3.1. Construct Ω_n by considering all possible scenarios of sub-batches to retrieve the target batch of containers in node n. Compute the probability of each node n_i in Ω_n.

Step 3.2. Apply *Abstract*(.) to each node n_i in Ω_n. If the abstract configuration is new, add n_i to Ω_n^{APBFS} and compute $f(n_i)$. If the abstract configuration is identical to a node m that is already in Ω_n^{APBFS}, add the probability of n_i to the probability of m.

Step 3.3. Compute $f(n)$ by summing up the expectation of $f(n_i)$ for each n_i, $n_i \in \Omega_n^{APBFS}$.

Step 4. If n is a SD node, compute $f(n)$ following Step 4.1~4.6.

Step 4.1. Construct Φ_n by considering all feasible retrieval sequences to retrieve the target batch of containers in node n.

Step 4.2. Apply *Abstract*(.) to each node n_i in Φ_n to avoid duplicate configurations, which leads to Φ_n'.

Step 4.3. Compute the lower bound $lb(n_i)$ for each node n_i in Φ_n'. Sort the nodes in Φ_n' in non-decreasing order of $lb(.)$.

Step 4.4. Compute $f(n_{(1)})$, wherein $n_{(1)}$ is the node with the smallest lower bound in Φ_n', and add $n_{(1)}$ to Φ_n^{APBFS}.

Step 4.5. Repeat for each of the remaining nodes in Φ_n' in non-decreasing order of $lb(.)$ to construct Φ_n^{APBFS}: If the lower bound of the considered node $n_{(k)}$ is less than the smallest $f(.)$ of the nodes in Φ_n^{APBFS}, apply *Abstract*(.) to node $n_{(k)}$. If the abstract configuration is identical to a node m that is already in the decision tree, then add m to Φ_n^{APBFS}; otherwise, add the considered node to Φ_n^{APBFS} and compute the cost-to-go function of the considered node $f(n_{(k)}) = APBFS(n_{(k)}, LB\text{-}FS)$.

Step 4.6. Determine $f(n)$ by taking the minimal value of $f(n_i)$, $n_i \in \Phi_n^{APBFS}$.

Step 5. If n is a RD node, compute $f(n)$ following Step 5.1~5.6.

Step 5.1. Construct Δ_n by considering all feasible relocation moves to retrieve the target container in node n.

Step 5.2. Apply *Abstract*(.) to each node n_i in Δ_n to avoid duplicate configurations, which leads to Δ_n'.

Step 5.3. Compute the lower bound $lb(n_i)$ for each node n_i in Δ_n'. Sort the nodes in Δ_n' in non-decreasing order of $lb(.)$.

Step 5.4. Compute $f(n_{(1)})$, wherein $n_{(1)}$ is the node with the smallest lower bound in Δ_n', and add $n_{(1)}$ to Δ_n^{APBFS}.

Step 5.5. Repeat for each of the remaining nodes in Δ_n' in non-decreasing order of $lb(.)$ to construct Δ_n^{APBFS}: If the lower bound of the considered node $n_{(k)}$ is less than the smallest $f(.)$ of the nodes in Δ_n^{APBFS}, apply *Abstract*(.) to node $n_{(k)}$. If the abstract configuration is identical to a node m that is already in the decision tree, then add m to Δ_n^{APBFS}; otherwise, add the considered node to Δ_n^{APBFS} and compute the cost-to-go function of the considered node $f(n_{(k)}) = APBFS(n_{(k)}, LB\text{-}FS)$.

Step 5.6. Determine $f(n)$ by taking the minimal value of $f(n_i)+r(n)$, $n_i \in \Delta_n^{APBFS}$.

4.5 The extended APBFS algorithm for the Sooo extension model

In this section, the APBFS algorithm is extended to solve the Sooo extension model optimally. The basic idea of the extended APBFS algorithm is, for each SD node n, to find its best offspring that minimizes the secondary objective among its offspring that minimize the primary objective, i.e., $f(n)$. Because of Observation 1 in Section 3.2.2, the secondary objective can be substituted by a waiting time indicator. With Observation 1, we define $\sum_{i \in B_k} r_i^{\zeta_k} \cdot \left(C_k - o_i^{\zeta_k} + 1\right)$ as the ***waiting time indicator*** of batch k under ζ_k, which is jointly determined by the container retrieval sequence in batch k and the number of relocations for each retrieval in batch k. Recalling in the APBFS algorithm, the container retrieval sequence of batch k is included in the immediate offspring (RD node) of the SD node n ($\lambda_n = C - L_k + 1$), denoted by node m. Given such a RD node m, the optimal number of relocations for each retrieval from batch k can be obtained by tracing the series of optimal offspring of node m. Therefore, in the extended APBFS algorithm, the focus is on selecting the optimal immediate offspring of SD nodes by using the waiting time indicator. To this end, the APBFS is extended from three perspectives. First, the pruning strategy is adjusted to explore each candidate node that is promising with regard to the secondary objective. Specifically, for each SD node n, its offspring whose lower bounds are equal to the best $f(n)$ found so far are not pruned (Step 4.5 in Algorithm 3). Second, for the immediate offspring of SD node n whose cost-to-go functions are equal to $f(n)$, we compute their waiting time indicators and then choose the one with the minimum waiting time indicator as the best offspring of node n (Step 4.7-4.8 in Algorithm 3). For a SD node n, the waiting time indicator of its immediate offspring n_i, denoted by $w(n_i)$, is given by *WaitTimeIndic*(n_i, n) in Algorithm 2. Lastly, the decision tree is traversed to level one, i.e., $\lambda^* = 1$ (Step 2 of Algorithm 3), because our *LB-FS* does not apply to the secondary objective.

Algorithm 2. Waiting time indicator of the immediate offspring n_i of SD node n: $w(n_i) = WaitTimeIndic(n_i, n)$

Step 1. Set C_n to be the number of containers in the target batch in node n.

Step 2. Set $m = n_i$ and $w(n_i) = 0$.

Step 3. For j from 1 to C_n, do Step 3.1~3.3.

Step 3.1. Set $r(m)$ to be the number of blocking containers in node m.

Step 3.2. Add $r(m) \cdot (C_n - j + 1)$ to $w(n_i)$.

Step 3.3. Update m by letting the new m become the optimal offspring of the current m.

For the purpose of completeness, the steps of the extended APBFS algorithm for computing $f(n)$ given a configuration n and lower bound *LB-FS* are presented below, and a sample decision tree developed by the extended APBFS algorithm is provided and explained in Appendix A.2. The differences from the APBFS algorithm are highlighted in bold font. Note that the $f(n)$ returned by the APBFS algorithm and the extended APBFS algorithm is exactly the same, but the best offspring of SD node n may be different due to the consideration of the secondary objective function.

Algorithm 3. Extended APBFS algorithm $f(n) = Extended\ APBFS(n, LB\text{-}FS)$

Step 1. Identify the class of node n. If n is a SD node, go to Step 4. Otherwise, go to Step 2.

Step 2. If the level of n is not greater than **one, return zero to** $f(n)$. Otherwise, go to Step 3.

Step 3. If n is a chance node, compute $f(n)$ following Step 3.1~3.3.

Step 3.1. Construct Ω_n by considering all possible scenarios of sub-batches to retrieve the target batch of containers in node n. Compute the probability of each node n_i in Ω_n.

Step 3.2. Apply *Abstract*(.) to each node n_i in Ω_n. If the abstract configuration is new, add n_i to Ω_n^{APBFS} and compute $f(n_i)$. If the abstract configuration is identical to a node m that is already in Ω_n^{APBFS}, add the probability of n_i to the probability of m.

Step 3.3. Compute $f(n)$ by summing up the expectation of $f(n_i)$ for each n_i, $n_i \in \Omega_n^{APBFS}$.

Step 4. If n is a SD node, compute $f(n)$ and **return the optimal offspring of *n* following Step 4.1~4.8.**

Step 4.1. Construct Φ_n by considering all feasible retrieval sequences to retrieve the target batch of containers in node n.

Step 4.2. Apply *Abstract*(.) to each node n_i in Φ_n to avoid duplicate configurations, which leads to Φ_n'.

Step 4.3. Compute the lower bound $lb(n_i)$ for each node n_i in Φ_n'. Sort the nodes in Φ_n' in non-decreasing order of $lb(.)$.

Step 4.4. Compute $f(n_{(1)})$, wherein $n_{(1)}$ is the node with the smallest lower bound in Φ_n', and add $n_{(1)}$ to Φ_n^{APBFS}.

Step 4.5. Repeat for each of the remaining nodes in Φ_n' in non-decreasing order of $lb(.)$ to construct Φ_n^{APBFS}: If the lower bound of the considered node $n_{(k)}$ is **not greater than** the smallest $f(.)$ of the nodes in Φ_n^{APBFS}, apply *Abstract*(.) to node $n_{(k)}$. If the abstract configuration is identical to a node m that is already in the decision tree, then add m to Φ_n^{APBFS}; otherwise, add the considered node to Φ_n^{APBFS} and compute the cost-to-go function of the considered node $f(n_{(k)}) = Extended\ APBFS(n_{(k)}, LB\text{-}FS)$.

Step 4.6. Determine $f(n)$ by taking the minimal value of $f(n_i)$, $n_i \in \Phi_n^{APBFS}$.

Step 4.7. **For each node in Φ_n^{APBFS}, compute its waiting time indicator by Algorithm 2.**

Step 4.8. **Return the node in Φ_n^{APBFS} that has the minimum waiting time indicator as the optimal offspring of node *n*.**

Step 5. If n is a RD node, compute $f(n)$ and **return the optimal offspring of *n* following Step 5.1~5.7**.

Step 5.1. Construct Δ_n by considering all feasible relocation moves to retrieve the target container in node n.

Step 5.2. Apply *Abstract*(.) to each node n_i in Δ_n to avoid duplicate configurations, which leads to Δ_n'.

Step 5.3. Compute the lower bound $lb(n_i)$ for each node n_i in Δ_n'. Sort the nodes in Δ_n' in non-decreasing order of $lb(.)$.

Step 5.4. Compute $f(n_{(1)})$, wherein $n_{(1)}$ is the node with the smallest lower bound in Δ_n', and add $n_{(1)}$ to Δ_n^{APBFS}.

Step 5.5. Repeat for each of the remaining nodes in Δ_n' in non-decreasing order of $lb(.)$ to construct Δ_n^{APBFS}: If the lower bound of the considered node $n_{(k)}$ is less than the smallest $f(.)$ of the nodes in Δ_n^{APBFS}, apply *Abstract*(.) to node $n_{(k)}$. If the abstract configuration is identical to a node m that is already in the decision tree, then add m to Δ_n^{APBFS}; otherwise, add the considered node to Δ_n^{APBFS} and compute the cost-to-go function of the considered node $f(n_{(k)})$ = *Extended APBFS*($n_{(k)}$, *LB-FS*).

Step 5.6. Determine $f(n)$ by taking the minimal value of $f(n_i)+r(n)$, $n_i \in \Delta_n^{APBFS}$.

Step 5.7. **Return the node in Δ_n^{APBFS} whose cost-to-go function equals $f(n)$ as the optimal offspring of node *n*.**

5. Heuristic solution methods

The (extended) APBFS algorithms are very time-consuming for larger problems. In this section, we propose two efficient heuristic algorithms for the SCRP-FS: the SEM (Sequencing based Expected Minmax) heuristic and the SEML (Sequencing based Expected Minmax with Look-ahead horizon) heuristic. In addition, in order to make the results of the Sooo (extension) model comparable to that of the batch model proposed by Galle et al. (2018b) in terms of the influence of service policies, we extend the EM (Expected Minmax) heuristic used in Galle et al. (2018b) to solve the batch model in the new context with customer preference-based arrivals. From now on, we use the "base model" to refer to the batch model that considers the customer preference-based arrivals. The contributions of our heuristics are summarized below.

First, the EM extension heuristic generalizes the EM heuristic to the SCRP with customer preference information. The main adaptation we made to the EM heuristic is the introduction of the concepts of the Blocking Index (BI) and the Delay Index (DI) that calculate the stack score. The BI and DI are not needed in the EM heuristics and they cannot be easily inferred from the case of equal arrival probability. As presented in Appendix B.1, great effort has been made to calculate the BI and DI, which are used to make a more accurate decision in case of a tie in the context of non-equal arrival probability. This extension is especially useful for the situation of large batch size as the occurrence of ties will be more frequent.

Second, we develop two new fast and efficient heuristics to solve the SCRP-FS – the SEM heuristic and the SEML heuristic. The main ingredients of these two heuristics are: sequencing rule and relocation rule. Regarding the relocation rule, we derive a new blocking index and a new delay index – BIS and DIS, to calculate the stack score by considering the SOOO policy, as shown in Appendix B.3. This generalizes the EM extension heuristic to a more flexible case. In addition, we make a contribution in terms of the sequencing rule, which is an important element of the SCRP-FS. Even though the SEM heuristic uses an intuitive sequencing rule, it has been shown to be effective in the computational experiments. The SEML heuristic further improves the SEM by using a more complex sequencing rule that applies a look-ahead strategy dedicated to performing the most promising retrieval sequence.

We present these three heuristics respectively in the following three sub-sections.

5.1 EM extension heuristic

The EM heuristic has been computationally demonstrated to be the equal best heuristic for the batch model. The idea of the EM heuristic is from that of the Min-Max heuristic in the earlier literature (Caserta et al., 2012), which is based on the computation of a stack score that determines which stack a blocking container should be placed. In this paper, the EM extension heuristic is adapted from the EM heuristic to obtain sub-optimal solutions of the base model. Although the EM extension heuristic is not the focus of this study, its description provides a basis for explaining the SEM heuristic and the SEML heuristic that we will design to solve the SCRP-FS.

Before describing the EM extension heuristic, we first briefly introduce the EM heuristic for the batch model. In the batch model, once the truck arrival order of a batch is revealed, the retrieval sequence for this batch is confirmed. The EM heuristic only focuses on the heuristic rules for relocating. Let l_c be the priority label of container *c* to be relocated, and $m(s)$ be the smallest label of a container in stack *s*, $s \in \{1,...,S\}$. For an empty stack, $m(s)$ is defined as *C*+1. The heuristic rules that determine the storage position of a blocking container *c* from stack *s* are described below.

[Condition 1] There is an available stack $s' \neq s$ such that $m(s') > l_c$.

Let $M = \min_{s' \in \{1,...,S\} \setminus s} \{m(s') : m(s') > l_c\}$. Select the stack that satisfies $m(s') = M$. Break ties by choosing from the highest ones, finally selecting the leftmost one if any ties remain.

[Condition 2] For all stack $s' \neq s$, $m(s') \leq l_c$.

Let $M = \max_{s' \in \{1,...,S\} \setminus s} \{m(s')\}$. Select the stack that satisfies $m(s') = M$. Break ties by choosing from the ones with the minimum number of containers labeled *M*. Further ties are broken by choosing from the highest ones, finally selecting the leftmost one if any ties remain.

The EM heuristic is relatively intuitive. The idea is to minimize the expected number of blocking containers after each relocation move. In Condition 1, *c* can surely avoid being relocated again, and we say a 'good' move is possible. In this condition, EM chooses the stack with the minimum $m(s')$, since the stacks with larger $m(s')$ can be saved as candidate stacks for positioning blocking containers with greater labels. In condition 2, we say a 'good' move is impossible. There are two cases. If $M = l_c$ (which implies that *c* will be relocated again in the future with probability), the stack with the minimum number of containers labeled *M* is chosen, which can minimize the probability of *c* being relocated again. The rationale behind it is that there is an equal chance for any container being the first one to be retrieved among the containers labeled *M*. On the other hand, if $M < l_c$ (which means that *c* will surely be relocated again in the future), the stack with the maximum $m(s')$ is chosen to delay the next relocation of *c* as much as possible. Ties are broken by selecting the stack with the minimum number of containers labeled *M* to delay *c* being relocated again, as there is an equal chance for any container being the first one to be retrieved among containers labeled *M*.

Now we extend the EM as an application to the base model. The EM extension follows the heuristic rule for Condition 1 but applies new rules for Condition 2. The following rules are used in the EM extension for Condition 2:

Let $M = \max_{s' \in \{1,...,S\} \setminus s} \{m(s')\}$. Select the stack that satisfies $m(s') = M$. In case of ties, if $M = l_c$, choose from the ones with the minimum $BI(s')$; if $M < l_c$, choose from the ones with the minimum $DI(s')$. Further ties are broken by choosing from the highest ones, finally selecting the leftmost one if any ties remain.

The main difference between the EM and the EM extension is the way of breaking ties in the case where more than one stack satisfies $m(s') = M$ in Condition 2. With the consideration of customer preference, more accurate criteria are required to break the tie. For this purpose, we introduce two indexes to calculate the stack stores: Blocking Index (BI) and Delay index (DI). The BI of a stack *s*, denoted by $BI(s)$, is defined as the probability of a container being blocking if relocated to *s*. The DI of a stack *s*, denoted by $DI(s)$, is defined as the probability of a container with the smallest label in stack *s* being the first one to be retrieved within its batch. The details of how to calculate the BI and the DI are given in Appendix B.1.

5.2 SEM (Sequencing based Expected Minmax) heuristic

The SEM heuristic is proposed to solve the SCRP-FS. Two decisions are to be made by the SEM: sequencing the trucks within the same sub-batch, and relocating the blocking containers. The main idea of the sequencing rule is to avoid as many current relocations as possible. The relocating rule is similar to the EM extension heuristic but new blocking index and delay index are introduced to consider the SOOO policy. In the following, we first introduce the outline of the SEM

heuristic and then describe the heuristic rules in detail.

5.2.1 Outline of the SEM

In the SEM heuristic, the decision on the retrieval sequence is made one container at a time using a sequencing rule and then the consequent blocking containers are relocated using a relocating rule. The following notations are defined and used throughout Section 5.2 and 5.3:

t_i : the *i*th target container, $i \in \{1,...,C\}$.

r_c : the number of relocations needed for retrieving container *c*.

X : the bay configuration. Let X_0 represent the initial configuration.

lmin : the smallest label of containers in X.

Θ : the set of containers labeled *lmin* in X.

The general steps of the SEM heuristic are as follows:

Step 0. Let $\mathrm{X} = \mathrm{X}_0$. Set *k*=1 and *i*=1, i.e., the index of the first batch and the index of the first target container.

Step 1. If *k* > *K*, STOP – all containers have been retrieved; otherwise, given X and the truck arrival information of batch *k*, update X by adding the number of containers in the first sub-batch to the labels of the containers in the second sub-batch.

Step 2. Identify *lmin* and construct Θ. If there is only one container in Θ, let this container be t_i; otherwise, determine t_i and update X accordingly using the ***Sequencing Rule***.

Step 3. Calculate r_{t_i}. If $r_{t_i} = 0$, go to step 4; otherwise, move the r_{t_i} number of blocking containers from top to bottom to the stacks determined by the ***Relocating Rule*** and X is updated as a result.

Step 4. Retrieve t_i from X. If $i = \sum_{j=1}^{k} C_j$, which means all containers in batch *k* have been retrieved, then set *k*=*k*+1 and go to step 1; otherwise, set *i*=*i*+1, go to step 2.

5.2.2 Heuristic rules in the SEM

The sequencing rule and the relocation rule are introduced here.

Sequencing rule

The SEM heuristic uses an intuitive sequencing rule, the main idea of which is choosing the container with the least number of blocking containers from the candidate containers.

Step 1. Given X, *lmin*, and Θ, compute the r_c of each container $c \in \Theta$.

Step 2. Sort $\{r_c : c \in \Theta\}$ in non-decreasing order of r_c. Choose the one with the lowest r_c from Θ as the target container t_i, breaking ties arbitrarily.

Step 3. Update X by increasing the labels of the containers in $\Theta \setminus t_i$ by one.

Appendix B.2 provides an example illustrating the above sequencing rule.

Relocating rule

The relocating rule used in the SEM heuristic follows the basic idea of the rule in the EM extension heuristic but uses a new blocking index and a new delay index - BIS (blocking index considering sequencing) and DIS (delay index considering sequencing) - to break ties. This is important because the blocking container in the batch model is not necessarily blocking in the SCRP-FS in which the container retrieval sequence is flexible. Therefore, in order to make correct decisions for the relocation positions, we need new indexes that can take into account the flexible service sequence. The idea behind the BIS is that container *c* being blocking if relocated to stack s' occurs only in the scenario where *c* is in the latter sub-batch and there is at least one container $c_i \in M_{s'}$ in the former sub-batch, where $M_{s'}$ is the set of

containers labeled *M* and located in s'. The idea behind the DIS is that a container c_i is surely being the first one to be retrieved in its batch only in the situation that satisfies the following two conditions: 1) c_i is in the former sub-batch; 2) c_i has the lowest number of blocking containers among the containers in the former sub-batch. The details of computing $BIS(s)$ and $DIS(s)$ are given in Appendix B.3.

For the sake of completeness, the relocating rule of the SEM to determine the storage position of a blocking container *c* from stack *s* is presented as follows.

[Condition 1] There is an available stack $s' \neq s$ such that $m(s') > l_c$.

Let $M = \min_{s' \in \{1,\ldots,S\} \setminus s} \{m(s') : m(s') > l_c\}$. Select a stack that satisfies $m(s') = M$. Break ties by choosing from the highest ones, finally selecting the leftmost one if any ties remain.

[Condition 2] For all stack $s' \neq s$, $m(s') \leq l_c$.

Let $M = \max_{s' \in \{1,\ldots,S\} \setminus s} \{m(s')\}$. Select a stack that satisfies $m(s') = M$. In case of ties, if $M = l_c$, choose from the ones with the minimum $BIS(s')$; if $M < l_c$, choose from the ones with the minimum $DIS(s')$. Further ties are broken by choosing from the highest ones, finally selecting the leftmost one if any ties remain.

5.3 SEML (Sequencing based Expected Minmax with Look-ahead horizon) heuristic

The SEML improves the SEM by using a sophisticated sequencing rule that applies a look-ahead strategy dedicated to performing the most promising retrieval sequence. Recalling the sequencing rule of the SEM heuristic, in case of tie that more than one container has the lowest number of blocking containers among the containers with the smallest labels, i.e., there is more than one potential target container, the SEM chooses one arbitrarily as the next target container (Step 2 in Section 5.2.2). The idea of the SEML heuristic is to break this tie more precisely with look-ahead evaluation. The look-ahead horizon *H* is equal to the number of potential target containers in case of the tie. To be more specific, the SEML first evaluates the contribution of each feasible retrieval sequence of the potential target containers to the total number of relocations, and then, the sequence that contributes least is selected as the actual retrieval sequence of these potential target containers. The contribution is measured by the sum of the number of realized relocations during the retrievals of the potential target containers and the lower bound of the configuration after these retrievals. A new lower bound is proposed with minor modification of the *LB-FS*, because the SEML applies to both SD nodes and RD nodes while the *LB-FS* does not apply to SD nodes. In the new lower bound, the containers with the same priority label whose truck arrival sequence have been revealed are not considered blocking each other. The relocating rule used in the SEML is the same as that in the SEM. The details of the sequencing rule are presented below, and an illustration is provided in Appendix B.4.

Step 1. Given X, *lmin*, and Θ, compute the r_c of each container $c \in \Theta$.

Step 2. Sort $\{r_c : c \in \Theta\}$ in non-decreasing order of r_c. Construct the set of potential target containers $\{c \mid r_c = \min\{r_c : c \in \Theta\}\}$. Set $H = \left|\{c \mid r_c = \min\{r_c : c \in \Theta\}\}\right|$. If *H*=1, then choose the only potential target container as the target container t_i. Update X by increasing the labels of the containers in $\Theta \setminus t_i$ by one. Otherwise, go to Step 3.

Step 3. Look-ahead evaluation

Step 3.1. Update X by increasing the labels of the non-potential containers in Θ by *H*.

Step 3.2. Enumerate all feasible retrieval sequences (*H*! number in total) for the potential target containers.

Step 3.3. Update the labels of the potential target containers according to one feasible retrieval sequence that has not been evaluated and obtain a tentative configuration to be evaluated.

Step 3.4. Given the tentative configuration, retrieve the potential target containers, move the blocking containers according to the relocating rule, and count the number of relocations incurred. Compute the lower bound

of the consequent configuration and the contribution. If all retrieval sequences have been evaluated, choose the one with the least contribution as the determined retrieval sequence for the potential target containers, breaking ties arbitrarily. Then, update X according to the determined retrieval sequence of the potential target containers; otherwise, go to Step 3.3.

Step 3.5. The container with $lmin$ is selected as the target container t_i.

6. Simulation model

In this section, we develop a discrete-event simulation model to evaluate the effectiveness of the exact algorithms and the heuristics respectively in terms of the two performance metrics: the total number of relocations and the average relevant truck waiting time. Simulation is needed for evaluating heuristics because the solutions of the heuristics depend on the scenario of truck arrivals. The necessity of a simulation model for evaluating exact algorithms is because the exact algorithms to be evaluated (the (extended) APBFS algorithm and the PBFS algorithm) do not record the service completion time for each batch. Hence, in order to evaluate the relevant truck waiting time, we need to simulate the complete retrieval process by using the optimal solutions. To the best of the authors' knowledge, this study is the first one that implements a simulation model to evaluate SCRP's optimal solutions that are derived from in a decision tree. Our main focus in this section is to show how to evaluate the solutions of the exact algorithms by using the developed discrete-event simulation model.

6.1 Input and output data

The input data of the simulation model includes: i) the problem instance that consists of the container stacking configuration, the batch information and the customer preference, ii) the truck arrival times, and iii) the handling time per relocation move and the handling time per retrieval move. The direct output for each container/truck includes: i) the number of relocations, ii) the service starting time, and iii) the service completion time. Then, we can output the total number of relocations and the average relevant truck waiting time. By definition, the relevant waiting time of a truck is calculated by: service completion time – service starting time – the handling time per retrieval move. The average relevant waiting time for a sample is obtained by taking the average over the relevant waiting times of all trucks. In addition, we also output the average delay and the average turn time, which will be explained in Section 7.3.2.

6.2 Model structure and functions

The simulation model consists of three major programs: a truck generator, an optimizer, and a simulator, which are subsequently described in detail. All programs are implemented in Matlab.

6.2.1 Truck generator

The truck generator program creates truck arrival times. Given a problem instance with an initial priority matrix and a customer matrix, N samples of truck arrival times are generated by respecting the appointed time windows and customer preferences. First, the sub-batch of a truck is generated based on the probability given by its customer preference p, such that on expectation each truck is allocated to the first sub-batch for $N*p$ times and the second sub-batch for $N*(1-p)$ times. Second, the sub arrival time window of the truck is generated by using its sub-batch, its appointed time window, and the length of the implemented appointment time window. Last, the specific arrival time of the truck is uniformly generated within its sub arrival time window. To ensure a fair comparison of different algorithms, seed initialized distribution is used. Thus, identical random truck arrival times can be used in simulating different algorithms and the simulation results are repeatable by applying identical problem instances.

6.2.2 Optimizer and simulator

The optimizer program generates the decisions on retrieval sequence and relocation positions, which feeds the simulator to perform tasks. The simulator is the core of the simulation model. Its main task is to perform the moves specified by the output of the optimizer, keep track of the state of the container stack, count the number of relocations, and record the time-related performance. The simulation model can evaluate both the exact algorithms and the heuristics but differs in

the optimizers and the way the simulators extract the decisions from the output of the optimizers. When evaluating heuristics, the relevant heuristic is used as an optimizer; and the simulator reads both the problem instance and the output of the truck generator. When evaluating the exact algorithms, the exact algorithm is used as an optimizer to produce the optimal solutions; and the simulator reads the problem instance and executes the optimal solutions. Details of the simulation model for evaluating exact algorithms are described below.

The simulation model contains three types of discrete events: revealing the truck arrival information for a batch, relocating a container, and retrieving a container. Given a problem instance, first, the optimizer is invoked, that is, an exact algorithm is executed to obtain the optimal solution. The obtained optimal solution is cached in a tree structure, which we call 'solution tree'. The simulator reads a truck arrival sample output by the generator and reveals it batch by batch. Once a batch is revealed, the simulator looks up the solution tree to extract the decisions for that batch and performs retrieval moves and relocation moves accordingly. An overview of the architecture of the simulation model is presented in Fig. 6.

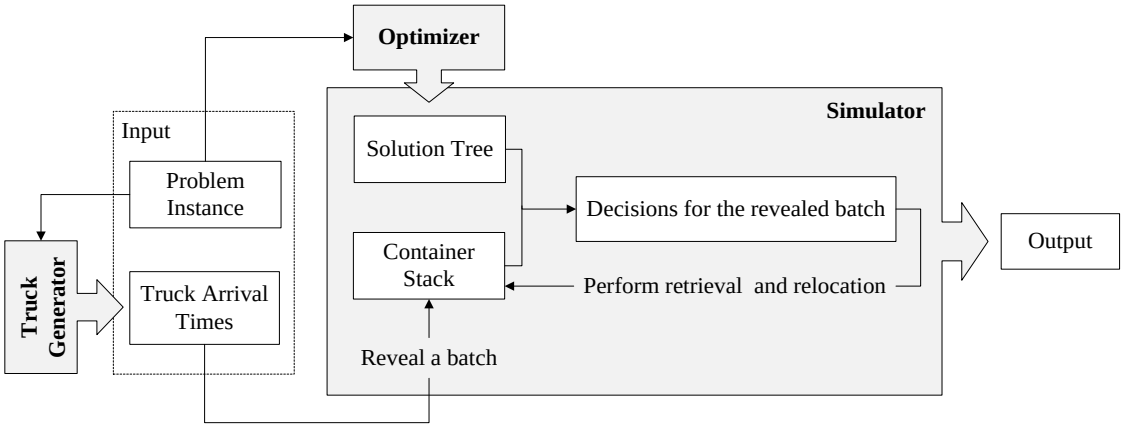


Fig. 6 The architecture of the simulating model for evaluating exact algorithms

We use the example in Fig. A.2 in Appendix A to illustrate the simulation process. Given the problem instance, i.e., the initial node in Fig. A.2, the optimizer is first invoked to generate the solution tree. Then, given a sample of truck arrival times, the simulator reads the sample and reveals it batch by batch. Once a batch is revealed, the simulator looks up the solution tree to identify the SD that matches the revealed container stack and extracts the optimal retrieval sequence for that batch. The decision on the retrieval sequence of a batch is included in the best offspring (a RD node) of the identified SD node. For example, let us consider the scenario in which the truck arrival information of the first batch is revealed as that in node n. Firstly, the optimal retrieval sequence for the first batch is extracted to be the one indicated in node n_2. The simulator then retrieves container 3 and records its service start time and service completion time. Secondly, the container stack is changed to n_3. Notice that the next container to be retrieved (container 4) has a blocking container. When there are blocking containers to be relocated, the simulator looks up the solution tree to identify the optimal relocation positions. The decisions on the relocation positions are obtained by tracing the best offspring (e.g., n_4) of the node in which the blocking containers are located (e.g., n_3). In the considered scenario, the best relocation position for the blocking container above container 4 is identified to be the empty stack. The simulator then relocates the blocking container to the empty stack, retrieve container 4, and records the service starting time and service completion time of container 4. Finally, the container stack is changed to node n_4. After that, the simulator continues to reveal the truck arrival information for the second batch and perform tasks in the same way. It should be noted that after each event, the container stack needs to be abstracted to ensure that it can be matched with one of the nodes in the solution tree.

7. Computational experiments

In this section, we test the proposed models and solution methods through enormous numerical experiments using the simulation model introduced in Section 6. We present four sets of experiments. Firstly, we test the solving capabilities of the two proposed exact solution algorithms; and we show the improvement of the Sooo extension model over the Sooo

model on the relevant truck waiting time. Secondly, we evaluate the effectiveness and the efficiency of the two proposed heuristics by comparing them with the exact solutions of the Sooo extension model; and we compare the performances of the two heuristics to conclude a superior one. Thirdly, we evaluate the effect of the proposed flexible service policy as opposed to the FCFS policy and analyze the impacts of the combinations of different bay layouts and fill rates, average batch sizes and customer preferences on the effect of the flexible service policy. Lastly, we analyze the influence of customer preference on the Sooo extension model.

All algorithms and simulation models are coded in MATLAB 2018a, partially based on the source code of Galle et al. (2018b) which is available at https://github.com/vgalle/StochasticCRP. All experiments are performed on a desktop with Intel® Core ™ i5-7500 3.40 GHz CPU, 8 GB of RAM, and 64-bit Windows 10 Enterprise. The time limit for running the exact algorithm for each instance is set to one hour (3600 seconds) because some instances are extremely time-consuming.

Our experiment dataset is adapted from the set of CRPTW instances in the literature (Ku and Arthanari, 2016a) which is available at http://crp-timewindow.blogspot.com. The existing instance set is composed of 1440 instances forming 48 classes. The problem classes are characterized by the size ($T\times S$) and the fill rate (μ) of the bay, with T varying from three to six tiers, S varying from five to ten stacks and two μ being considered: 50% and 67%. Given a bay size and a fill rate, the number of containers in the bay is calculated by C = round(μ*T*S), where round(x) rounds x to its closer integer. There are on average two containers per batch, i.e., the average batch size is two. For each such class setting, 30 instances are included, varying in the stacking positions of the containers and the number of containers of each batch. To provide a meaningful interpretation for our model, we consider larger batches with up to on average six containers per batch. The instances of larger batches are obtained by slightly modifying the existing instance set following the method in Galle et al. (2018b), which merges r batches using $w' = \lceil w/r \rceil$, where w is the original batch of a container and w' is its modified batch. As a result, we have instances with small batches (on average 2 containers per batch), large batches (on average 4 containers per batch), and ultra-large batches (on average 6 containers per batch). We use 'the number of tiers (T) – the number of stacks (S) – the fill rate (μ) – the average batch size (B)' to represent our *problem class*. We do not distinguish problem scales accurately because all the relevant factors – T, S, μ, B – have an influence on the computation times of the exact algorithms and the random initial configuration of the container stack also has a great influence. Instead, we consider a problem as a larger problem if it has a larger rate and/or larger batches while other factors (T and S) are the same. Because the instances with ultra-large batches are very hard to be solved optimally, we only use their near-optimal solutions obtained by heuristics to show a positive difference between the FCFS policy and the SOOO policy in Section 7.3.2.

Regarding the customer preference, we consider three scenarios of homogenous preference, in which the preferences of all trucks are respectively 0%, 50%, and 100%, and a scenario of heterogeneous preference, in which the preference of each truck is randomly generated and thus differs from each other. The instances with scenario '50%' are referred to as the benchmark set, as they are equivalent to the instance set of the batch model. In this scenario, the probability of each SD node is the same (i.e., 0.25). In the scenario '0%' and '100%', all trucks will arrive at the second sub-time window and the first sub-time window respectively. The truck generator program introduced in Section 6.1 is used to generate 1000 samples of truck series for each instance associated with a scenario of customer preference. The appointment time window is set to be 30 minutes. The handling times per relocation move and per retrieval move are calibrated according to the technical capabilities of yard cranes. A Rubber-Tyred Gantry Crane (RTG) can perform 20–25 moves on average per hour, but its realized performance in practice is typically less than 12 moves per hour (Saanen, 2011), which indicates that each move takes on average 2.4 - 5 minutes. Considering that a retrieval move usually takes longer to complete than a relocation move due to the need for coordination with the truck drivers, we set 2 minutes for a relocation move and 4 minutes for a retrieval move.

7.1 Performance of the proposed models and exact algorithms

In this section, we evaluate the performances of the exact algorithms, test the tightness of *LB-FS*, and compare the average relevant truck waiting time between the Sooo model and the Sooo extension model. All results are obtained by simulating optimal solutions.

7.1.1 Performances of the exact algorithms

Table 2 shows the results of instances with small batches and the 50% fill rate. The first three columns list the problem class, which is characterized by the number of stacks (*S*), the number of tiers (*T*), and the number of containers (*C*). The average batch size is omitted here as all instances have the same average batch size (i.e., two). Column 'lb' gives the value of lower bound obtained by our proposed lower bound *LB-FS*. Columns five to nine and Columns ten to fourteen respectively report the simulation results of the APBFS algorithm and the extended APBFS algorithm. Column 'Opt' gives the average of the expected total number of relocations over 30 instances, which is the theoretically optimal solution obtained by the exact algorithm. Colum 'Solved' reports the number of instances that the relevant exact algorithm is able to solve to optimality within the time limit (1 hour), where '√' indicates that all 30 instances for a problem class are solved to optimality. Column 'CPU(s)' reports the average computation time for the solvable instances in seconds. Column 'Rel' and column 'AveWait' respectively give the average of the total number of relocations and the average of the relevant truck waiting time over 30 instances, each one based on 1000 samples, which are obtained by simulation.

Table 2 Results of the APBFS and the extended APBFS algorithm for instances with small batches and 50% fill rate

T	*S*	*C*	lb	Sooo - APBFS					Sooo extension - extended APBFS				
				Opt	Solved	CPU(s)	Rel	AveWait (min)	Opt	Solved	CPU(s)	Rel	AveWait (min)
3	5	8	1.454	1.478	√	0.02	1.478	3.319	1.478	√	0.03	1.478	3.207
	6	9	1.558	1.582	√	0.02	1.581	3.428	1.582	√	0.03	1.581	3.303
	7	11	2.608	2.654	√	0.02	2.654	3.485	2.654	√	0.03	2.654	3.356
	8	12	2.163	2.169	√	0.01	2.169	3.091	2.169	√	0.03	2.169	3.014
	9	14	2.875	2.884	√	0.02	2.885	3.226	2.884	√	0.04	2.885	3.106
	10	15	3.092	3.094	√	0.03	3.093	3.044	3.094	√	0.06	3.093	2.930
4	5	10	2.725	2.856	√	0.02	2.855	3.534	2.856	√	0.03	2.855	3.411
	6	12	3.371	3.461	√	0.03	3.466	3.532	3.461	√	0.05	3.466	3.351
	7	14	3.817	3.944	√	0.04	3.941	3.485	3.944	√	0.05	3.941	3.324
	8	16	4.475	4.555	√	0.16	4.559	3.556	4.555	√	0.20	4.559	3.390
	9	18	5.467	5.526	√	0.29	5.523	3.691	5.526	√	0.34	5.523	3.474
	10	20	5.983	6.016	√	0.72	6.015	3.334	6.016	√	0.80	6.015	3.181
5	5	13	4.371	4.883	√	0.16	4.883	4.042	4.883	√	0.19	4.884	3.900
	6	15	5.138	5.546	√	2.44	5.544	3.708	5.546	√	2.98	5.544	3.553
	7	18	6.246	6.575	√	0.72	6.573	3.993	6.575	√	0.77	6.573	3.777
	8	20	7.017	7.519	√	7.57	7.516	3.695	7.519	√	8.16	7.516	3.482
	9	23	8.358	8.699	29	67.35	8.696	3.835	8.699	29	68.25	8.696	3.606
	10	25	8.883	9.237	29	39.40	9.238	3.719	9.237	29	49.37	9.238	3.517
6	5	15	5.975	7.004	√	4.56	6.999	4.034	7.004	√	5.06	6.999	3.886
	6	18	6.900	7.729	√	5.48	7.728	4.162	7.729	√	6.18	7.728	3.959
	7	21	8.575	8.925	23	294.96	8.991	4.026	8.925	22	160.87	8.923	3.787
	8	24	9.258	9.886	22	150.07	9.881	3.998	9.886	22	164.07	9.882	3.748
	9	27	10.275	10.538	18	100.24	10.538	3.661	10.538	18	104.47	10.538	3.419
	10	30	11.692	11.576	19	285.31	11.599	3.785	11.576	18	108.52	11.576	3.580

*Note: customer preference scenario: 50%

From Table 2, we can see that the solution capacity of the two algorithms is quite similar. Both of them are capable of solving all the instances with *T*= 3 and *T*=4 in less than one second and 98.9% (178/180) of the instances with *T*=5 within the time limit. As *T* increases to six, some instances are extremely time-consuming. We call the instances that cannot be solved within one hour 'hard instances'. The number of hard instances for each problem class is basically the same as that for the PBFS algorithms, which indicates that our proposed exact algorithms are effective for the SCRP-FS. We observe that the hard instances for the two proposed algorithms are the same except the classes of (*T*=6, *S*=7) and (*T*=6, *S*=10).

This is because the search rules used in the two algorithms are basically the same. The only difference is that in the extended APBFS algorithm the nodes that perform equally in terms of the primary objective are further explored in order to find the node that is optimal to the secondary objective. This leads to that the extended APBFS algorithm requires a longer CPU time to prove optimality, which can be observed from the CPU columns. It should be pointed out that for the problem classes of (T=6, S=7) and (T=6, S=10), the CPU times for Sooo extension are much shorter than that of Sooo. This is because the Sooo model is able to solve one more instance than the Sooo extension model within the allowed computational time limit (i.e. 3600 seconds), and this extra instance is too time-consuming to solve for the Sooo extension model. A fair comparison of two models can be referred to Section 7.1.2 and Table C.4. Due to the unavailability of the optimal solutions for hard instances, these hard instances are excluded from the simulation. For the problem class that includes any hard instance, Table 2 only reports the average over the solved instances in columns 'CPU', 'Rel' and 'AveWait'. In addition, as expected, the optimal solutions in terms of the total expected number of relocations (Opt) of the two models are the same. Besides, the gap between the 'Opt' and the 'Rel' in both models is insignificant, which is within [-0.08%, 0.09%], indicating that our samples are large enough to approximate the actual values.

The results of the extended APBFS algorithm for larger instances are given in Appendix C.1. From Table 2 and Appendix C.1, we can conclude that the extended APBFS can solve 87.5% (42/48) of the instances with T=3,4 within 30 seconds. In the tables in Appendix C.1, lb* represents the calibrated lower bound, which takes the average of the lb of the instances that are solved optimally. By comparing lb*and Opt, we can find that the relative difference between the lower bound and the optimal solution for instances with higher stacks (larger T) is greater than that with lower stacks (smaller T). This can be explained by the fact that the chance of a container being relocated more than once in a bay with higher stacks is greater than that in a bay with lower stacks. Since our lower bound only counts the number of blocking containers that are relocated at least once, it is tighter for lower bays. For all the instances with lower bays (T=3, 4) in Table 2 and Appendix C.1, our lower bound is within 13.11% of the optimal solution, and in about 73% (35/48) cases our lower bound is very close to the optimal solution with a gap within 5%. Since in many container terminals, laden containers are stacked up to four tiers due to safety issues and efficiency considerations, our lower bound can efficiently evaluate the least number of relocations needed to empty a bay, which could help to determine a favorable container stacking configuration.

Remark: it is observed that the CPU time deviates greatly for different instances even their problem classes are the same. After a closer check, we find that the initial configuration of the container stack has a great influence on the solution computational efficiency.

7.1.2 Comparison of the Sooo model and the Sooo extension model

For a fair comparison of the relevant truck waiting time between the two models, we calibrate the results of 'CPU', 'Rel' and 'AveWait' in Table 2 to ensure that only the instances that are solved optimally by both algorithms are included into the comparison. The calibrated results are presented in italic in Table C.4 of Appendix C.2. Besides, we compare the results of the two models for instances with a 67% fill rate, which are shown in Table 3 and Table 4 (the numbers in italic represent the calibrated results). Comparing the 'Rel' of the two models, it is found that the simulated total number of relocations of the two models are the same in most cases, with only five problem class occurring a difference of 0.001 number of relocations. The occurrence of this difference seems counterintuitive, but it might happen only because we are sampling. However, as the difference is quite trivial, it is fair to compare the 'AveWait' of the two models based on our samples. Column 'Gap[AveWait]' reports the gap between the average relevant truck waiting time of the two models, which represents the benefits of taking into account the truck waiting time.

Comparing Table 3 with Table C.4, we can find that the gaps in the average relevant truck waiting time between the two models are more significant for instances with a larger fill rate. Besides, in a comparison between Table 3 and Table 4, it can be found that the instances with more concentrated truck arrival patterns, that is, '0%' customer preference scenario, benefit more from the Sooo extension model. In addition, the results of small batches and large batches indicate that the batch size does not have a significant influence on the relative difference of the relevant truck waiting time between the two models. From Table C.4 and Tables 3-4, we can conclude that the reduction in the average relevant truck

waiting time in the Sooo extension model over the Sooo model is between 2.5% and 11%. Moreover, the CPU results confirm that the extended APBFS algorithm takes a longer time to obtain the optimal solution than the APBFS algorithm.

Table 3 Comparison between the Sooo model and the Sooo extension model for instances with 67% fill rate and '50%' customer preference scenario

T	*S*	*C*	Sooo				Sooo extension				Gap [AveWait]
			Solved	CPU(s)	Rel	AveWait (min)	Solved	CPU(s)	Rel	AveWait (min)	
Small batches											
3	5	10	√	0.04	2.786	3.387	√	0.05	2.786	3.224	4.81%
	6	12	√	0.04	3.620	3.710	√	0.05	3.620	3.524	5.01%
	7	14	√	0.03	3.759	3.655	√	0.06	3.759	3.506	4.08%
	8	16	√	0.05	4.382	3.577	√	0.08	4.382	3.385	5.37%
	9	18	√	0.07	4.816	3.580	√	0.11	4.816	3.408	4.80%
	10	20	√	0.07	5.066	3.327	√	0.11	5.066	3.176	4.54%
4	5	13	√	0.11	5.071	4.130	√	0.14	5.071	3.976	3.73%
	6	16	√	1.10	6.931	4.109	√	1.19	6.931	3.898	5.14%
	7	19	√	1.01	6.929	3.646	√	1.16	6.929	3.465	4.96%
	8	21	√	13.98	7.969	3.829	√	20.25	7.969	3.601	5.95%
	9	24	√	11.08	9.257	3.825	√	11.80	9.257	3.619	5.39%
	10	27	27	69.81	9.622	3.620	27	92.17	9.622	3.426	5.36%
Large batches											
3	5	10	√	0.51	2.403	7.208	√	0.90	2.403	6.945	3.65%
	6	12	√	0.58	3.245	7.804	√	1.21	3.245	7.468	4.31%
	7	14	√	1.41	3.518	8.119	√	3.58	3.518	7.720	4.91%
	8	16	√	1.13	4.180	7.671	√	3.36	4.179	7.281	5.08%
	9	18	√	1.46	4.484	7.680	√	4.05	4.485	7.349	4.31%
	10	20	√	0.93	4.755	7.338	√	3.40	4.755	6.937	5.46%
4	5	13	√	8.60	4.632	8.618	√	27.60	4.631	8.342	3.20%
	6	16	28	13.31	6.167	8.502	28	39.35	6.167	8.091	4.83%
	7	19	28	36.99	6.307	7.681	28	175.06	6.307	7.310	4.83%
	8	21	28	*12.32*	*7.027*	*8.036*	26	73.15	7.027	7.580	5.67%
	9	24	27	*71.98*	*8.208*	*8.086*	24	320.01	8.208	7.655	5.33%
	10	27	23	*153.87*	*8.898*	*7.673*	21	441.13	8.898	7.233	5.73%

Table 4 Comparison between the Sooo model and the Sooo extension model for instances with 67% fill rate and '0%' customer preference scenario

T	*S*	*C*	Sooo				Sooo extension				Gap [AveWait]
			Solved	CPU(s)	Rel	AveWait (min)	Solved	CPU(s)	Rel	AveWait (min)	
Small batches											
3	5	10	√	0.03	2.500	3.307	√	0.04	2.500	2.993	9.48%
	6	12	√	0.03	3.367	3.672	√	0.04	3.367	3.289	10.44%
	7	14	√	0.03	3.567	3.595	√	0.05	3.567	3.329	7.42%
	8	16	√	0.05	4.100	3.546	√	0.07	4.100	3.171	10.58%
	9	18	√	0.07	4.533	3.533	√	0.10	4.533	3.204	9.33%
	10	20	√	0.07	4.867	3.320	√	0.10	4.867	3.023	8.94%
4	5	13	√	0.06	4.600	3.944	√	0.09	4.600	3.667	7.02%
	6	16	√	0.18	6.600	4.067	√	0.21	6.600	3.633	10.66%
	7	19	√	0.20	6.567	3.589	√	0.25	6.567	3.239	9.77%
	8	21	√	1.06	7.633	3.803	√	1.31	7.633	3.384	11.02%
	9	24	√	2.96	8.967	3.836	√	3.26	8.967	3.422	10.79%
	10	27	29	7.71	9.379	3.630	29	8.56	9.379	3.246	10.56%
Large batches											
3	5	10	√	0.46	1.800	6.887	√	0.83	1.800	6.327	8.13%
	6	12	√	0.50	2.700	7.683	√	1.07	2.700	6.861	10.70%
	7	14	√	1.23	2.967	7.990	√	3.48	2.967	7.152	10.49%
	8	16	√	0.94	3.533	7.471	√	2.33	3.533	6.704	10.26%
	9	18	√	1.08	3.833	7.452	√	2.88	3.833	6.852	8.05%
	10	20	√	0.82	4.233	7.287	√	2.99	4.233	6.503	10.75%
4	5	13	√	1.82	3.667	8.103	√	6.19	3.667	7.441	8.16%
	6	16	√	8.28	5.467	8.283	√	132.93	5.467	7.433	10.26%
	7	19	√	10.48	5.533	7.484	√	173.77	5.533	6.789	9.28%
	8	21	√	*9.28*	*6.214*	*7.762*	28	86.46	6.214	6.986	9.99%
	9	24	√	25.06	7.733	8.042	√	331.17	7.733	7.200	10.47%
	10	27	29	*27.63*	*8.429*	*7.569*	28	84.01	8.429	6.788	10.31%

7.2 Effectiveness of the proposed heuristics

In this section, we evaluate the effectiveness of the heuristic algorithms. First, we compare the results of the proposed two heuristics with that of the extended APBFS algorithm. Second, we compare the performances of the two heuristics.

7.2.1 Comparison of the exact solutions and heuristic solutions

Table 5 compares the SEM heuristic and the SEML heuristic with that of the extended APBFS algorithm on instances with small batches and a 50% fill rate. For a fair comparison, the heuristic results are calibrated (in italic) to ensure that the comparison is based on the instances that are solved to optimality by the extended APBFS algorithm. Because the CPU times of both heuristics are less than 1 second, they are not presented here. The best heuristic result for each problem class is highlighted in bold, from which it can be observed that in almost all cases the SEML heuristic outperforms the SEM heuristic. In addition, from the column Gap[Rel], we can see that for instances with T=3,4, our proposed SEML heuristic performs quite well, with only at most 0.64% difference from the optimal solutions in terms of the total number of relocations. Overall, the SEML heuristic is at most 3.41% more than the optimal total number of relocations. Regarding the average relevant truck waiting time (Gap[AveWait]), the result of the SEML heuristic is at most 1.49% more than that of the extended APBFS algorithm. Besides, occasionally, the SEML heuristic even outperforms the extended APBFS algorithm slightly in terms of the relevant truck waiting time. This is not surprising, because the Sooo extension model aims to minimize the total waiting times of each batch sequentially rather than minimizing the total waiting times of all

the trucks. Because the total waiting time of all trucks is jointly determined by the number of relocations of and the service sequence of each truck, the solutions with the same total number of relocations may lead to different total waiting time and it might also happen that the solutions with more relocations lead to less total waiting time.

Table 5 Comparison of the extended APBFS algorithm, SEM and SEML heuristics for small batches and 50% fill rate

T	*S*	*C*	Extended APBFS		SEM				SEML			
			Rel	AveWait	Rel	AveWait	Gap[Rel]	Gap[AveWait]	Rel	AveWait	Gap[Rel]	Gap[AveWait]
3	5	8	1.478	3.207	**1.478**	**3.208**	0.00%	0.02%	**1.478**	**3.208**	0.00%	0.02%
	6	9	1.581	3.303	1.582	**3.303**	0.07%	0.02%	**1.581**	**3.303**	0.00%	0.00%
	7	11	2.654	3.356	**2.654**	**3.359**	0.00%	0.08%	**2.654**	**3.359**	0.00%	0.08%
	8	12	2.169	3.014	2.174	3.015	0.22%	0.04%	**2.169**	**3.014**	0.00%	0.01%
	9	14	2.885	3.106	**2.886**	**3.106**	0.05%	0.01%	**2.886**	**3.106**	0.03%	0.01%
	10	15	3.093	2.930	**3.100**	**2.931**	0.22%	0.04%	**3.100**	**2.931**	0.22%	0.04%
4	5	10	2.855	3.411	2.891	3.408	1.26%	-0.09%	**2.874**	**3.403**	0.64%	-0.22%
	6	12	3.466	3.351	3.481	3.356	0.44%	0.15%	**3.471**	**3.353**	0.14%	0.06%
	7	14	3.941	3.324	3.986	3.332	1.15%	0.25%	**3.960**	**3.327**	0.49%	0.10%
	8	16	4.559	3.390	**4.578**	**3.389**	0.42%	-0.02%	**4.578**	**3.389**	0.42%	-0.02%
	9	18	5.523	3.474	**5.532**	**3.471**	0.15%	-0.09%	**5.532**	3.472	0.15%	-0.06%
	10	20	6.015	3.181	6.018	**3.183**	0.06%	0.06%	**6.016**	**3.183**	0.01%	0.06%
5	5	13	4.884	3.900	5.047	3.921	3.36%	0.55%	**5.031**	**3.920**	3.02%	0.51%
	6	15	5.544	3.553	5.633	3.558	1.61%	0.16%	**5.619**	**3.554**	1.35%	0.04%
	7	18	6.573	3.777	6.631	3.790	0.88%	0.33%	**6.619**	**3.785**	0.70%	0.20%
	8	20	7.516	3.482	7.636	3.500	1.60%	0.52%	**7.619**	**3.497**	1.37%	0.43%
	9	23	8.696	3.606	*8.733*	*3.610*	0.42%	0.11%	***8.710***	***3.607***	0.16%	0.04%
	10	25	9.238	3.517	*9.301*	*3.518*	0.68%	0.03%	***9.285***	***3.515***	0.50%	-0.05%
6	5	15	6.999	3.886	**7.237**	**3.944**	3.40%	1.48%	7.238	**3.944**	3.41%	1.49%
	6	18	7.728	3.959	7.918	3.964	2.47%	0.13%	**7.913**	**3.963**	2.40%	0.11%
	7	21	8.923	3.787	***9.011***	***3.772***	0.98%	-0.40%	***9.011***	***3.772***	0.98%	-0.40%
	8	24	9.882	3.748	***9.987***	***3.755***	1.06%	0.16%	*9.994*	***3.755***	1.13%	0.18%
	9	27	10.538	3.419	***10.652***	***3.429***	1.08%	0.28%	***10.652***	***3.429***	1.08%	0.28%
	10	30	11.572	3.580	***11.579***	***3.578***	0.06%	-0.06%	***11.579***	***3.578***	0.06%	-0.06%

*Note: customer preference scenario: 50%

We compare the SEML heuristic with the extended APBFS algorithm on larger instances in Appendix C.1. The comparisons are based on the instances that can be solved optimally by both the exact algorithm and the heuristic algorithm. It can be seen that for the problem classes in Table 5 and in Appendix C.1 for which we have access to the optimal solutions of all the 30 instances, the maximum gaps for the total number of relocations and the average relevant truck waiting time are 4.28% and 1.49% respectively; and in about 84% (41/49) cases, the total number of relocations obtained by the SEML heuristic is very close to the optimal solutions with gaps no more than 2%. Besides, for the problem classes for which we only have access to the optimal solutions of part of the 30 instances, the maximum gap for the total number of relocations and the average relevant truck waiting time are 9.80% and 1.53% respectively. With an enormous number of instances in a range of sizes being evaluated, our experiments show strong evidence that the SEML heuristic is a good solution to the SCRP-FS of practical sizes.

7.2.2 Comparison of the two heuristics

Furthermore, we compare the performance between the SME heuristic and the SMEL heuristic on all the instances with a 67% fill rate. Appendix D displays the gaps between the two heuristics for instances with three to six tiers respectively. The horizontal axis presents the characteristics of each instance: the customer preference scenario (*P*), the average batch size (*B*), and the number of stacks (*S*). Gap[Rel]=(SEM[Rel]–SEML[Rel])/SEM[Rel] × 100%, and Gap[AveWait]=

(SEM[AveWait]–SEML[AveWait])/SEM[AveWait] × 100%. In most cases, the SEML heuristic shows superior performance on both measures to that of SEM heuristic, which confirms the importance of looking ahead on the decision making of retrieval sequence. Although in very few cases the good performance on 'Rel' of the SEML heuristic is at the expense of 'AveWait', the increases on 'AveWait' are no more than 1% compared with SEM. Given the better performance quality of the SEML heuristic, we use SEML as the heuristic solver for the SCRP-FS in the remaining experiments.

7.3 Effect of the flexible service policy

In this section, we first verify the effectiveness of our proposed flexible service policy by comparing the Sooo extension model with the base model. Then, various instances with different bay sizes and fill rates, batch sizes and customer preference scenarios are tested to investigate their impacts on the effect of the flexible service policy.

7.3.1 Comparison of the base model and the Sooo extension model on the benchmark

In order to evaluate the effect of the proposed flexible service policy as opposed to the FCFS policy, we compare the results of the Sooo extension model with the base model on the benchmark set. The benchmark set consists of the instances with a 50% fill rate, small batches, and the '50%' customer preference scenario. In order to obtain the results of the base model, we slightly adapt the PBFS algorithm in Galle et al. (2018b) by using a new lower bound that incorporates the characteristics of customer preference, which is similar to the idea of computing the BI in the EM extension algorithm. Table 6 reports the calibrated results for comparison. Gap[Rel]=(Base model[Rel]–Sooo extension model[Rel])/ Base model[Rel]×100%, and Gap[AveWait]=(Base model[AveWait]– Sooo extension model[AveWait])/ Base model[AveWait]×100%. We can see that around 2% - 13% reduction in the total number of relocations can be achieved by the Sooo extension model compared with the base model on the benchmark set. The effectiveness of the flexible policy is also demonstrated by the around 4.3% - 8.4% reduction in the average relevant truck waiting time.

Table 6 Comparison between the base model and the Sooo extension model on the benchmark instance set

T	*S*	*C*	Base model*					Sooo extension model					Gap	
			Opt	Solved	CPU(s)	Rel	AveWait (min)	Opt	Solved	CPU(s)	Rel	AveWait (min)	Gap[Rel]	Gap [AveWait]
3	5	8	1.703	√	0.02	1.700	3.436	1.478	√	0.03	1.478	3.207	13.03%	6.66%
	6	9	1.739	√	0.01	1.737	3.487	1.582	√	0.03	1.581	3.303	9.00%	5.28%
	7	11	2.878	√	0.02	2.878	3.544	2.654	√	0.03	2.654	3.356	7.79%	5.30%
	8	12	2.308	√	0.02	2.307	3.148	2.169	√	0.03	2.169	3.014	5.95%	4.26%
	9	14	3.004	√	0.02	3.006	3.235	2.884	√	0.04	2.885	3.106	4.02%	3.98%
	10	15	3.192	√	0.02	3.193	3.066	3.094	√	0.06	3.093	2.930	3.14%	4.45%
4	5	10	3.108	√	0.02	3.107	3.657	2.856	√	0.03	2.855	3.411	8.08%	6.74%
	6	12	3.675	√	0.03	3.676	3.559	3.461	√	0.05	3.466	3.351	5.70%	5.85%
	7	14	4.164	√	0.03	4.164	3.522	3.944	√	0.05	3.941	3.324	5.37%	5.62%
	8	16	4.819	√	0.16	4.820	3.592	4.555	√	0.20	4.559	3.390	5.42%	5.63%
	9	18	5.730	√	0.33	5.729	3.670	5.526	√	0.34	5.523	3.474	3.58%	5.35%
	10	20	6.275	√	0.75	6.272	3.353	6.016	√	0.80	6.015	3.181	4.10%	5.13%
5	5	13	5.323	√	0.15	5.325	4.198	4.883	√	0.19	4.884	3.900	8.29%	7.09%
	6	15	5.911	√	4.38	5.914	3.813	5.546	√	2.98	5.544	3.553	6.25%	6.83%
	7	18	6.965	√	1.11	6.969	4.046	6.575	√	0.77	6.573	3.777	5.68%	6.64%
	8	20	7.847	√	7.85	7.846	3.703	7.519	√	8.16	7.516	3.482	4.21%	5.98%
	9	23	8.999	28	70.96	9.005	3.829	*8.704*	29	*70.61*	*8.701*	*3.609*	3.38%	5.76%
	10	25	9.547	29	47.92	9.547	3.716	9.237	29	49.37	9.238	3.517	3.24%	5.34%
6	5	15	7.595	√	4.98	7.590	4.244	7.004	√	5.06	6.999	3.886	7.79%	8.43%
	6	18	8.232	√	17.45	8.231	4.249	7.729	√	6.18	7.728	3.959	6.11%	6.82%
	7	21	*9.394*	23	*251.90*	*9.393*	*4.077*	8.925	22	160.87	8.923	3.787	5.00%	7.11%
	8	24	10.318	22	134.48	10.313	3.989	9.886	22	164.07	9.882	3.748	4.18%	6.03%
	9	27	10.713	18	103.13	10.714	3.609	10.538	18	104.47	10.538	3.419	1.64%	5.26%
	10	30	11.804	17	234.62	11.809	3.774	*11.551*	18	*114.81*	*11.547*	*3.583*	2.22%	5.06%

*Note: The base model refers to the batch model of Galle et al. (2018b) in the new context of customer preference-based

arrivals.

7.3.2 Effect of the flexible service policy in different scenarios

Based on all instances (including the instances with ultra-large batches), we analyze the impacts of the combinations of different bay sizes (T*S) and fill rates (μ), truck appointment patterns (the average batch size) and truck arrival behaviors (the customer preference scenario) on the effects of the flexible service policy. The results of all instances are obtained by simulating heuristic solutions except the results of the benchmark set which are from the optimal solutions in Table 6.

Effect on the number of relocations

Fig. 7 depicts the relative reduction in the total number of relocations. In each figure, six plots are presented, varying in the average batch size in the horizontal direction and the customer preference scenario in the vertical direction. Note that because the relative reductions for the '0%' and '100%' customer preference scenarios are the same, we only present the result of one scenario in the vertical direction. As shown in Fig. 7, the effect on relocation reduction is more significant for the cases with larger batch sizes and the cases with more concentrated truck arrivals within the appointed time window (i.e., the '0%' customer preference scenario). The reason is that these cases provide more opportunities for out-of-order retrievals to reduce relocations as there are more trucks in the same sub-batch. Note that under the cases where the customer preference scenario is '0%', the SCRP-FS is equivalent to the deterministic CRP with flexible service policies (CRP-FS) in which all the trucks in the same batch are allowed to be retrieved out-of-order. The effect of the flexible service policy is maximized in the context of the CRP-FS as the container retrieval order has the greatest flexibility and meanwhile, the truck arrival uncertainties are completely offset.

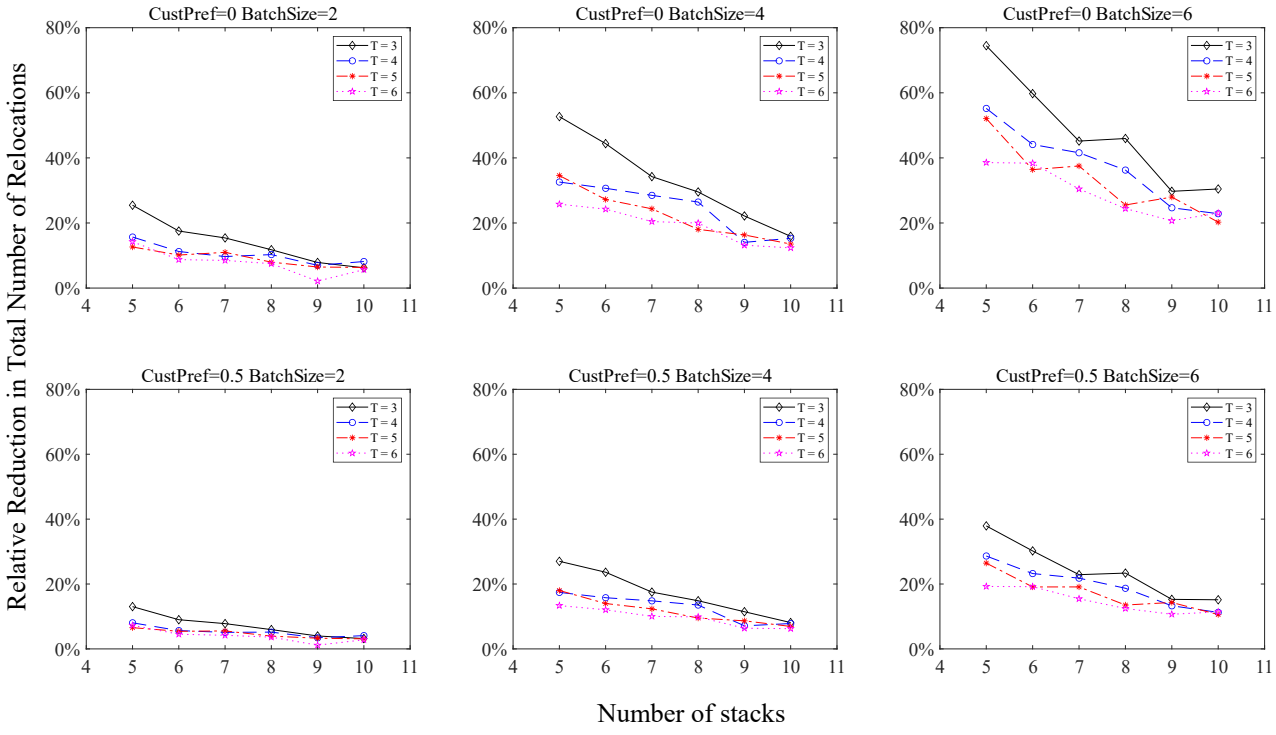


Fig. 7(a) Effect of the flexible service policy on the total number of relocations for instances of 50% fill rate

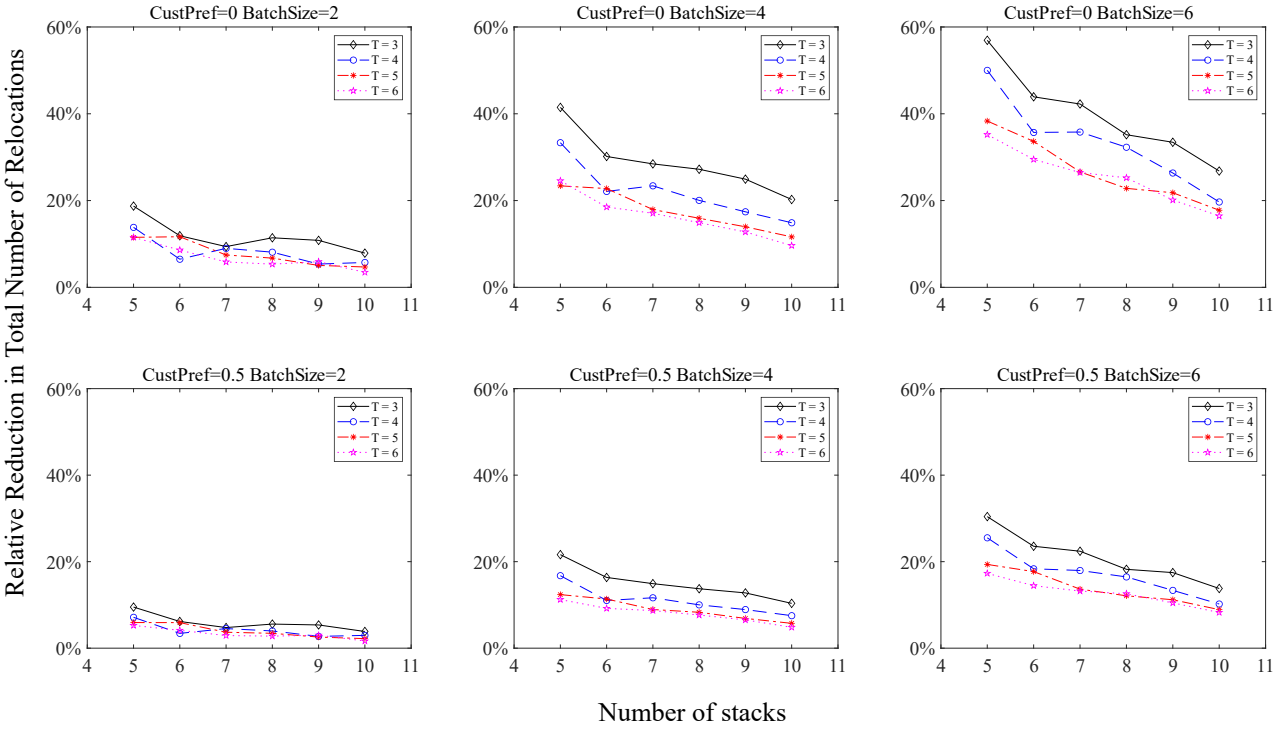


Fig. 7(b) Effect of the flexible service policy on the total number of relocations for instances of 67% fill rate

Furthermore, we can find that the relative reduction in the number of relocations depends on the bay size (T*S) and the fill rate (μ). In general, the percentage is decreasing as the bay size and the fill rate get larger. To understand this, let us consider the benefits of the flexible service policy. For each out-of-order retrieval, the direct benefit is avoiding one relocation, and the indirect benefit is avoiding future relocations that might be caused by the blocking container if it is not retrieved out-of-order. As T and μ increases, the likelihood of blocking become greater, but the increasing number of blocking containers cannot be offset completely by implementing the proposed flexible service policy as only the containers in the same sub-batch are allowed to be retrieved out-of-order. In addition, as S increases, it is more likely that a better relocating stack can be found for a relocated container, meaning that the relocated container being blocking again in the future is less likely to occur, and thus the benefit of out-of-order retrieval is diminishing. This indicates that the bay of smaller size and sparse stacking can benefit more from the flexible service policy. For the instances with on average six trucks per batch and the '50%' customer preference scenario, the peak relative reduction on the number of relocations is around 38% and 30% respectively for the bay of 50% and 67% fill rate. This leads to a 9.6% and 11.3% reduction in the average relevant waiting time respectively for the bay of 50% and 67% fill rate (see Appendix E.3).

Effect on the trucks waiting time

We also report the absolute reduction in the two performances. Note that the application of the flexible policy always leads to positive reductions, we use "absolute reduction" only to differentiate it from "relative reduction". The absolute reduction on the average relevant truck waiting times shows a similar pattern as that on the total number of relocations (see Appendix E.1 and Appendix E.2). However, in contrast to Fig. 7, the bay's height and fill rate has a positive impact on the relative reduction in the average relevant truck waiting time (see Appendix E.3). This is because the total relevant truck waiting times include a fixed amount of time that is not influenced by the service policy. Recalling Section 3.2.2, for each batch k, no matter what the solution is, we have to add $t^{ret} \cdot \sum_{j=1}^{C_k} (C_k - j)$ to the total relevant waiting times, which is a fixed value. To obtain an accurate understanding of the effect of the flexible service policy on reducing trucks' waiting times, we deduct this fixed amount of time from the total relevant waiting times and then take the average,

resulting in a new average waiting time. To differentiate, we call it average delay time. The average delay time represents the waiting time caused to each truck only due to relocation operations.

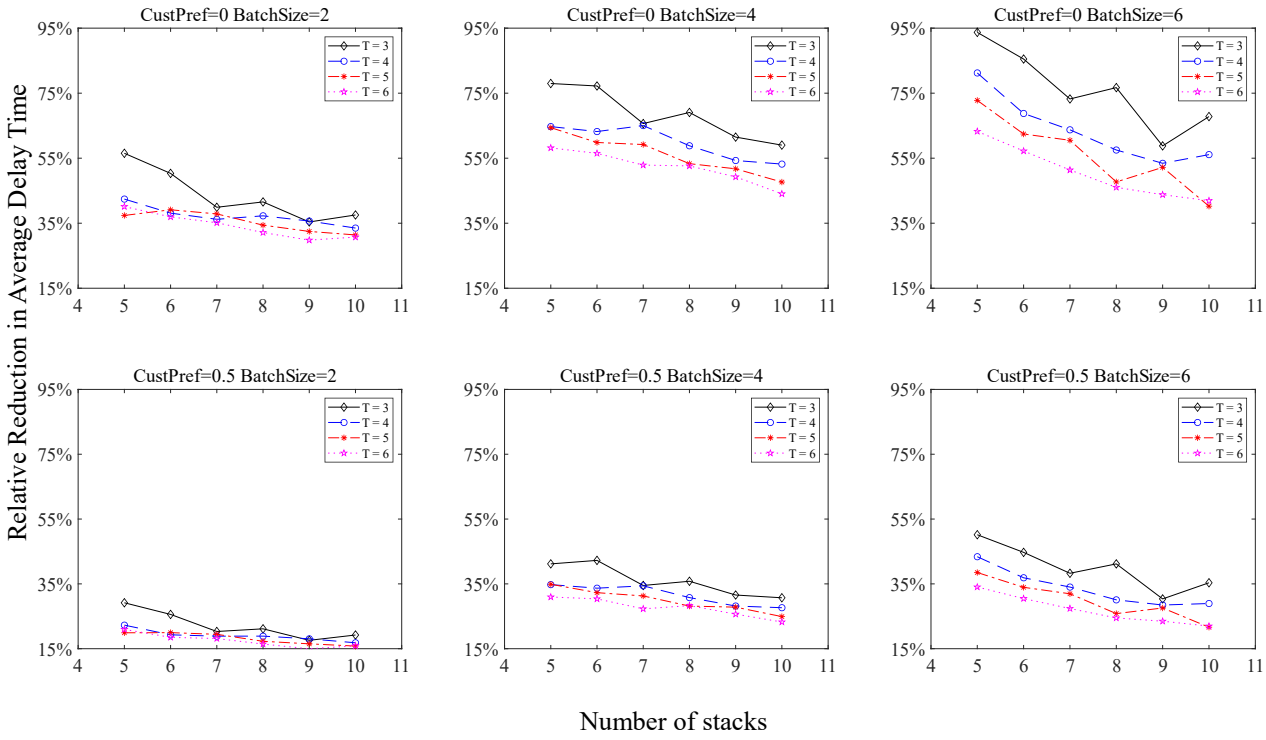


Fig. 8(a) Effect of the flexible service policy on average delay time for instances of 50% fill rate

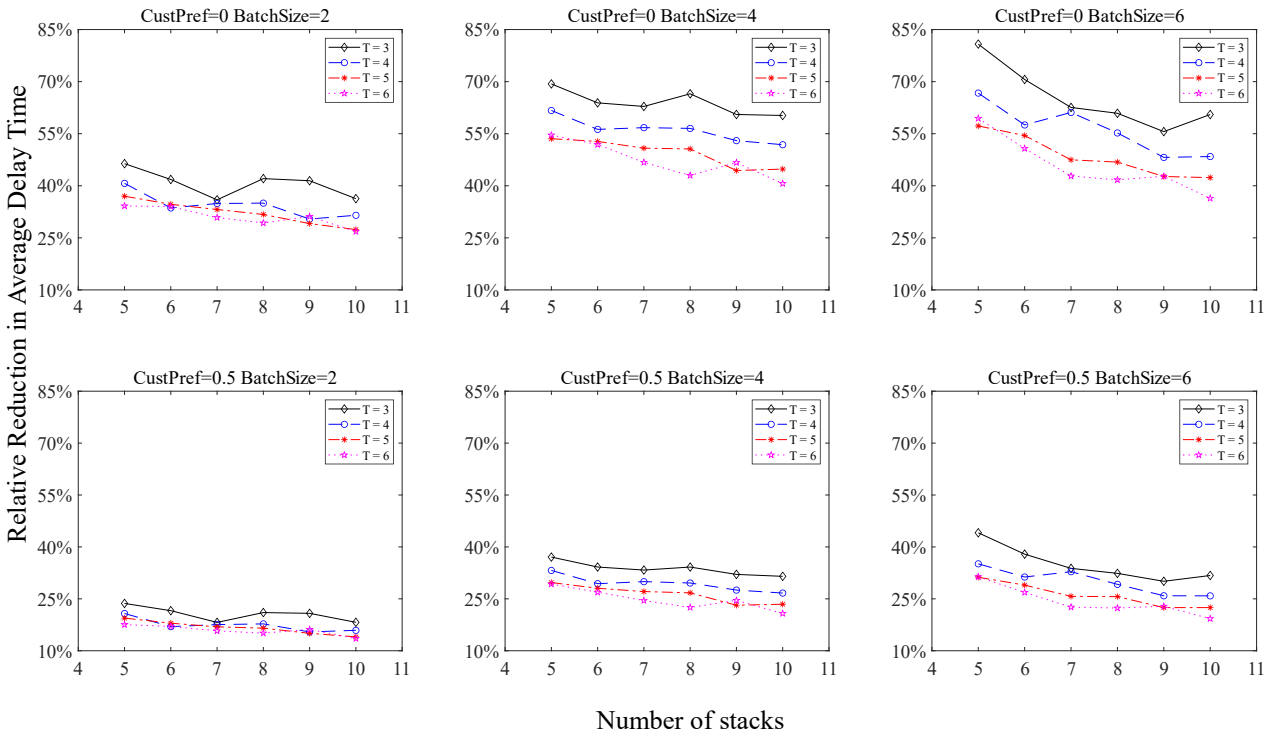


Fig. 8(b) Effect of the flexible service policy on average delay time for instances of 67% fill rate

Fig. 8 depicts the relative reduction in the average delay time. The similar trend between Fig. 7 and Fig. 8 indicates that the reduction in the number of relocations plays a direct role in reducing the average delay time. For the instances

with ultra-large batches and the customer preference scenario being '50%', about 50% and 44% of the average delay time can be reduced respectively for the bay of 50% and 67% fill rate as a result of out-of-order retrievals. The experiment results also demonstrate (not presented in the figure) that on average one reduction in the number of relocations results in 1.07 minutes and 0.93 minutes reduction in the average relevant truck waiting time across all instances respectively for the bay of 50% and 67% fill rate.

Moreover, we also measure the average turn time under the flexible service policy, which is shown in Appendix E.4. The turn time of a truck is defined as the elapse of time between its arrival time and its retrieval service completion time. Appendix E.4 shows an average difference of 15-minutes in the average turn time between the '0%' customer preference scenario and the '100%' customer preference scenario. This is only due to the difference between the truck arrival times that are generated for the two scenarios. Noticing that our appointment time window is set to be 30 minutes, the 15-minutes difference validates our simulation results.

Effect on the service equity

Out-of-order retrievals might make some trucks perceive unfair service due to the adjustment of the service sequence. To examine the equity of truck service, we use box plots to display the distributions of the truck turn time under the FCFS policy and the flexible policy respectively, which is contrasted in Fig. 9. It can be observed that the maximum values of the truck turn time (among all trucks' turn times including the outliers) under the flexible service policy are generally greater than that under the FCFS policy. This is not surprising because the flexible service policy makes some trucks that arrive earlier being served at a later time due to the sequencing decision. However, because we restrict the out-of-order retrievals within the same sub-group, the trucks arriving in the first sub-window will always be serviced before the trucks arriving in the second sub-window, which means the service equity between two sub-groups of trucks is maintained. It can be seen that the difference of the maximum turn times between two policies is only about five minutes among the cases in Fig. 9. Besides, the differences are not obvious for the cases with higher tiers ($T = 5, 6$), and in some cases, the flexible policy even has a shorter maximum turn time. The reason is that the instances with higher tiers require a higher average number of relocations to retrieve a container, while the flexible policy can significantly reduce the number of relocations and avoid the long waiting time compared to the FCFC service. Moreover, the flexible policy has a lower minimum value of the truck turn time; and more importantly, the median and the mean of the trucks' turn time under the flexible policy are always smaller than those under the FCFS policy.

These results demonstrate that when the FCFS policy is replaced by the flexible policy, although some trucks may experience a little longer turn time, on average the service each truck receives can be improved. This goal is consistent with most of the existing relevant literature, e.g., minimizing the average waiting time (Borjian et al., 2015b; Zeng et al., 2019) or minimizing total delay times (Borjian et al., 2013).

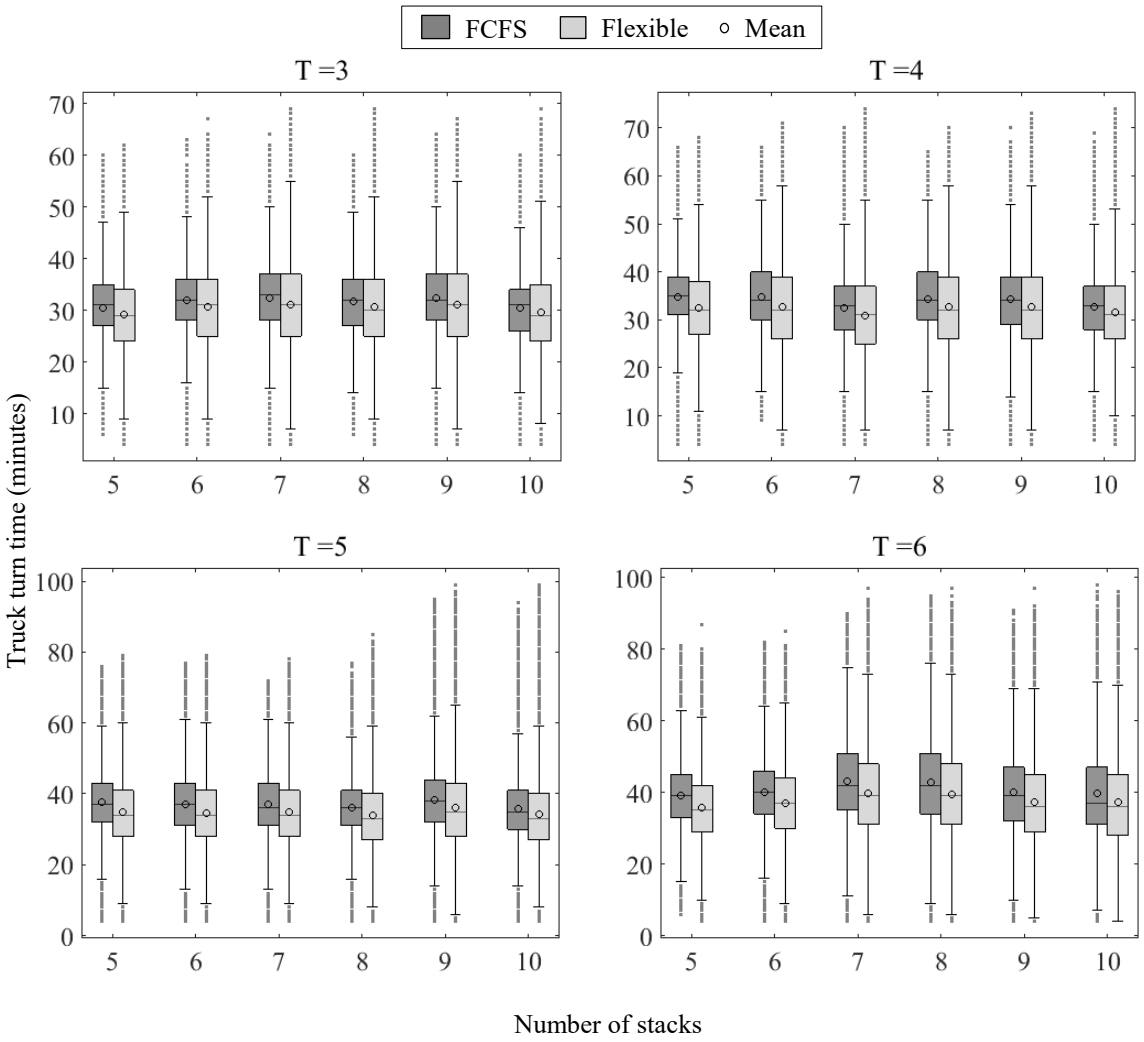


Fig. 9 Grouped box plots of the truck turn time under two service policies for the instances with 67% fill rate, '50%' customer preference scenario and on average 6 containers per batch

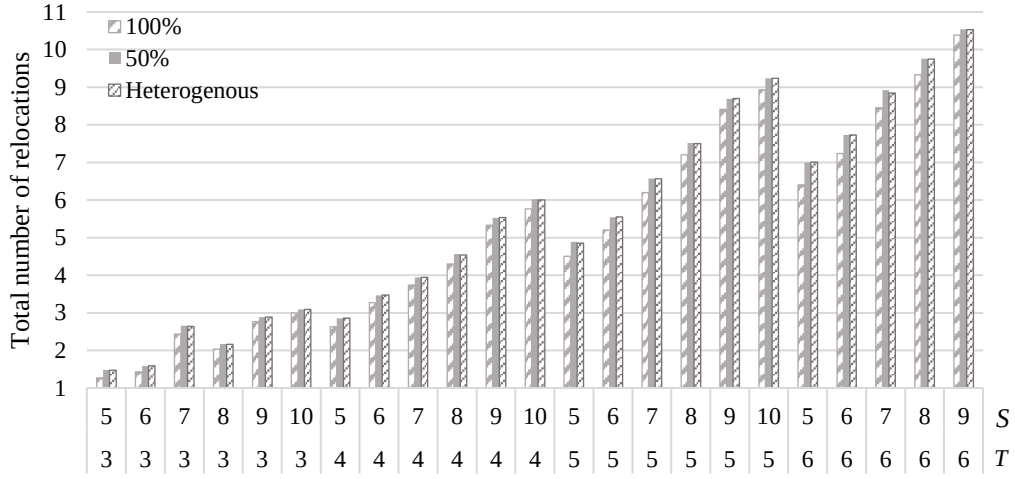
7.4 Influence of customer preference

In this section, we analyze the influence of customer preferences on the results of the Sooo extension model. We consider three sets of customer preference scenario: i) all trucks arrive at the first sub-time window with the probability of 100% ('100%'); ii) all trucks arrive at the first sub-time window with the probability of 50% ('50%'); iii) trucks arrive at the first sub-time window with different probabilities (heterogeneous). Appendix F reports the results obtained by the extended APBFS algorithm of these three sets of customer preference scenarios on the instances with small batches and a 50% fill rate. For the heterogeneous scenario, we generate 10 samples of customer preferences randomly for each of the 30 instances of each problem class, and hence, each problem class has 300 instances to be solved. The number of instances that are solved optimally is given in the form 'x/300' and '√' indicates that all instances out of 300 are solved optimally. Note that it takes about six days to obtain the results of the problem class with T=6 and S=9 for the heterogeneous scenario since 127 out of 300 instances cannot be solved optimally within one hour, we did not conduct the experiments of the problem class with T=6 and S=10 (because it would take much longer computational time than six days).

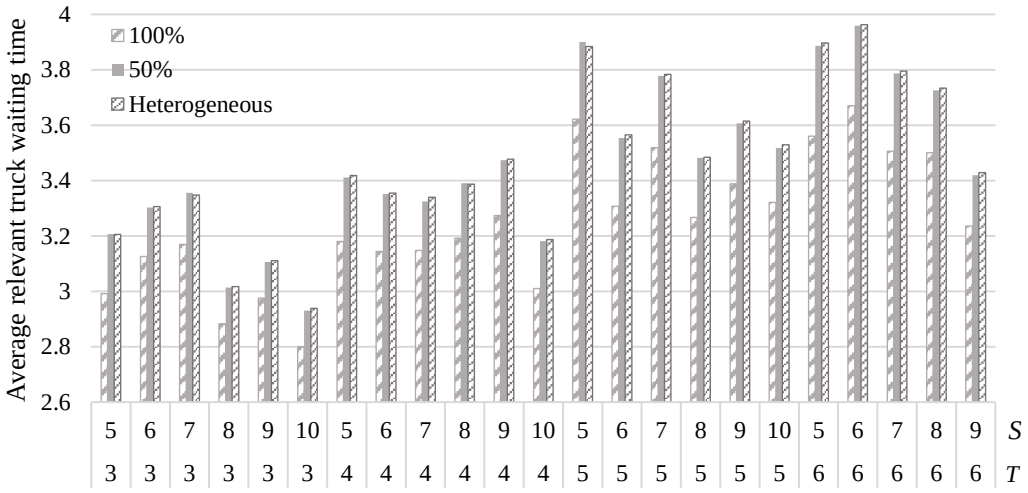
Concerning the computational efficiency, from Appendix F, it is observed that the instances with the '100%' scenario take less time to obtain optimal solutions than the '50%' scenario, and there are fewer hard instances for the '100%' scenario. This can be explained by the fact that in the '100%' scenario, each chance node only has one offspring, which reduces the burden of the decision tree. By contrast, the heterogeneous scenario takes more time to be solved. The reason is that the abstract technique does not work efficiently for the heterogeneous scenario as it rarely happens that two nodes

are equivalent since the preferences of customers differ from each other. Even so, there is no obvious change in the number of hard instances, which can be seen from the 'Solved' columns of the '50' scenario and the heterogeneous scenario.

In terms of the objective values, from Fig. 10, we can see that the heterogenous scenario and the '50%' scenario perform similarly, which have obvious differences from the '100' scenario. This implies that if all customers tend to arrive at a specific sub-time window of their appointed time windows, the results will be influenced significantly. Besides, if customers have heterogeneous preferences, we can use the results of the '50%' scenario as an approximation of the objective values of the heterogeneous scenario. However, this does not mean that the solution of the '50%' scenario is feasible to the solution of the heterogeneous scenario.



(a) Comparison of the total number of relocations



(b) Comparison of the average relevant truck waiting time

Fig. 10 Comparison between three sets of customer preference scenarios for instances with small batches and a 50% fill rate

We summarize the key findings of the experiments below. Firstly, the solution capacity of the two proposed exact solution algorithms is quite similar. The extended APBFS algorithm can solve 87.5% of the instances with T=3,4 within 30 seconds. Secondly, our proposed lower bound *LB-FS* is more effective for instances with lower tiers (T=3,4). In 73% of the instances with T=3,4, our lower bound is fairly close to the optimal solution with a gap within 5%. Thirdly, for the instances that can be solved optimally, the Sooo extension model can reduce the average relevant truck waiting time by 2.5% - 11% in comparison to the Sooo model, which indicates the significance of considering truck waiting time in addressing the SCRP-FS. Fourthly, the SEML heuristic outperforms the SEM heuristic in both performances, which demonstrates the importance of looking ahead on the decision-making of retrieval sequence. For the instances that we

have access to the optimal solutions, in about 84% cases the total number of relocations obtained by the SEML heuristic is very close to the optimal solutions with a gap within 2%, and the maximum gap is 4.28%. Fifthly, the proposed flexible service policy can significantly reduce both the number of relocations and the relevant truck waiting times. Although some trucks may experience a little longer turn time, on average the service each truck receives can be improved. For the benchmark instance set, the largest relative reduction on the number of relocations is around 38% and 30%, which leads to a 9.6% and 11.3% reduction in the average relevant waiting time, respectively, for the bay of a 50% and a 67% fill rate. The benefit is more obvious for the instances with smaller and sparse bays, larger batches, and concentrated truck arrivals within one of the sub-time windows. Lastly, customers preferring a specific sub-time window of their appointed time windows has a great influence on the results.

8. Conclusions

In this paper, we have considered the stochastic container relocation problem with flexible service policies (termed as SCRP-FS), which focuses on retrieving and relocating import containers with uncertain truck arrival orders. The trucks arrive at the terminal randomly within their appointed time windows. The containers whose designated trucks arrive at the same sub-time window are allowed to be retrieved out-of-order. Customers (trucks)' preference is taken into consideration to describe the randomness of truck arrivals within the same time window. The problem is first formulated by a stochastic dynamic programming model to minimize the expected number of relocations, which is termed as the Sooo model. Then a Sooo extension model is developed considering a primary objective the same as the Sooo model and a secondary objective of minimizing the total truck waiting times of each batch sequentially. The Sooo extension model not only considers the terminal operator's objective but also the trucks' objective. Built upon a state-of-the-art algorithm for solving the SCRP, tree search-based algorithms are developed to make optimal recommendations about the retrieval sequence of the next batch of containers and the relocation positions of the blocking containers. Moreover, two heuristic algorithms, SEM and SEML, are designed to seek high-quality solutions efficiently for practical-size problems. A discrete event-driven simulation model is developed to evaluate the performance of the algorithms (optimal and heuristic). Extensive computational experiments demonstrate the effectiveness of the models and the algorithms.

On the theoretical side, firstly, the SCRP-FS generalizes the conventional SCRP from two perspectives. On the one hand, the flexible service policy relaxes the traditional FCFS policy, which provides more opportunities for reducing the number of relocations and allows for reducing the trucks' waiting time as well. On the other hand, the assumption of uniformly distributed truck arrivals within the same time window is relaxed by a more general probabilistic model. The capability of capturing the customers' preference-based arrival behavior, in particular, is a major advantage of the probabilistic model. Secondly, the proposed methodology contributes to the literature of solving multiple objective multi-stage stochastic optimization problems by embedding the optimization of the secondary objectives within the multi-stage optimization procedure for the primary objective. Such methodology may be applicable to other transportation optimization problems such as berth allocation problems or train loading problems, in which decisions are made dynamically and multiple objectives are prioritized.

On the practical side, based on our findings, we provide some managerial insights to terminal operators and truck companies. Firstly, by slightly diverting the current FCFS service policy to the flexible service policy that implements out-of-order retrievals within half of the appointment time window, both the number of relocations and the average truck waiting time during the retrieval service can be significantly reduced; and the service equity between two sub-groups of trucks is maintained. Secondly, the flexible service policy is more beneficial in the following practical situations: the container terminal uses small bay or/and sparse stacking strategy; the containers to be retrieved in a bay are booked in large batches; the trucks arrive within either the earlier segment or the latter segment of their appointed time windows concentratedly. Thirdly, customer preference has a great influence on both the number of relocations and the truck waiting times during the retrieval service. Lastly, the developed SEML heuristic can generate good solutions very fast, which can be applied in practice to enable the real-time dynamic decision-making for the SCRP-FS.

This paper provides several directions for further research. Firstly, the proposed models and algorithms are reasonably

general and flexible, which allows for further refining and improvement, e.g. terminal operators could choose different sizes of the sub-batches or multiple sub-batches. The optimization framework will be similar and the structure of the decision tree does not need change. However, if the terminal operator decides to have more sub-batches, the size of the search tree will be larger due to the consideration of more possibilities of sub-batches. Hence, a more efficient search algorithm needs to be developed to obtain exact solutions. Nevertheless, our proposed heuristics are supposed to be still efficient because their time complexities are decreasing with the decrease of the batch size. Besides, the benefits of the flexible service policy need to be further evaluated because the terminal operator will have less control over the truck service sequence if they use more sub-batches. Secondly, this study could be extended to address more general SCRP problems where trucks do not necessarily arrive within their appointed time windows. In the real world, the arrival of a truck may be prior to or later than the appointed time window. An extended arrival time window that includes both the preceding and the succeeding time window relative to the appointed time window is more appropriate to predict trucks' arrival times. The probability of the deviation from the appointed time window could be gained from historical data and be considered as our proposed customer preference. Thirdly, based on our proposed lower bound, more efficient stacking policies considering the possibility of out-of-order retrievals in the future could be developed to stack import containers in an orderly configuration so that fewer relocations are needed during the retrieval process.

Acknowledgments

The authors thank three anonymous reviewers' constructive comments that have helped improve the presentation of the paper. We also thank Ku and Arthanari (2016a) and Galle et al. (2018b) for their published source data. This study is partially supported by the China Scholarship Council, the Royal Society (Grant No. IEC\NSFC\170100), the EU H2020 (Grant No. 777742, EC H2020-MSCA-RISE-2017), the National Natural Science Foundation of China (Grant No. 71671021), and Fundamental Research Funds for the Central Universities.

References

Bacci, T., Mattia, S. and Ventura, P., 2019. The bounded beam search algorithm for the block relocation problem. Comput. Oper. Res. 103, 252-264.

Bakker, H., Dunke, F. and Nickel, S., 2020. A structuring review on multi-stage optimization under uncertainty: Aligning concepts from theory and practice. Omega, 96, 102080.

Birge, J.R., Louveaux, F., 2011. Introduction to stochastic programming, Second Ed. Springer, New York, pp. 89.

Bonney, J. 2015. US ports move toward truck appointment model, JOC.com, Apr 27 2015. https://www.joc.com/port-news/us-ports/port-new-york-and-new-jersey/us-ports-move-toward-truck-appointment-model_20150427.html. (accessed. 23/02/2020).

Borjian, S., Manshadi, V., Barnhart, C., Jaillet, P., 2013. Dynamic Stochastic Optimization of Reshuffles in Container Terminals. Working paper, Massachusetts Institute of Technology, Cambridge.

Borjian, S., Galle, V., Manshadi, V.H., Barnhart, C., Jaillet, P., 2015a. Container Relocation Problem: Approximation, Asymptotic, and Incomplete Information. Working paper, Massachusetts Institute of Technology, Cambridge.

Borjian, S., Manshadi, V.H., Barnhart, C., Jaillet, P., 2015b. Managing Relocation and Delay in Container Terminals with Flexible Service Policies. Working paper, Massachusetts Institute of Technology, Cambridge.

Carlo, H.J., Vis, I.F.A., Roodbergen, K.J., 2014. Storage yard operations in container terminals: Literature overview, trends, and research directions. Eur. J. Oper. Res. 235, 412–430.

Caserta, M., Schwarze, S., Voß, S., 2012. A mathematical formulation and complexity considerations for the blocks relocation problem. Eur. J. Oper. Res. 219, 96–104.

Caserta, M., Schwarze, S., Voß, S., 2011a. Container Rehandling at Maritime Container Terminals, in: Böse, J.W. (Ed.), Handbook of Terminal Planning. Operations Research/Computer Science Interfaces Series. Springer, New York, pp. 247–269.

Caserta, M., Voß, S., Sniedovich, M., 2011b. Applying the corridor method to a blocks relocation problem. OR Spectr. 33, 915–929.

Chen, G., Govindan, K., Golias, M.M., 2013a. Reducing truck emissions at container terminals in a low carbon economy: Proposal of a queueing-based bi-objective model for optimizing truck arrival pattern. Transp. Res. Part E Logist. Transp. Rev. 55, 3–22.

Chen, G., Govindan, K., Yang, Z., 2013b. Managing truck arrivals with time windows to alleviate gate congestion at container terminals. Int. J. Prod. Econ. 141, 179–188.

Davies, P., 2009. Container Terminal Reservation Systems Paper. 3rd Annu. METRANS Natl. Urban Freight Conf. 1–19.

de Melo da Silva, M., Erdoğan, G., Battarra, M., Strusevich, V., 2018. The Block Retrieval Problem. Eur. J. Oper. Res. 265, 931–950.

Dragović, B., Tzannatos, E., Park, N.K., 2017. Simulation modelling in ports and container terminals: literature overview and analysis by research field, application area and tool. Flex. Serv. Manuf. J. 29, 4–34.

DP World. https://www.londongateway.com/port/book-a-vehicle (accessed. 28/02/2020).

Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, J.M., 2015. An exact approach for the Blocks Relocation Problem. Expert Syst. Appl. 42, 6408–6422.

fenixmarineservices.com. https://www.fenixmarineservices.com/terminal/#appointments. (accessed. 21/02/2020).

Galle, V., Barnhart, C., Jaillet, P., 2018a. A new binary formulation of the restricted Container Relocation Problem based on a binary encoding of configurations. Eur. J. Oper. Res. 267, 467–477.

Galle, V., Manshadi, V., Borjian Boroujeni, S., Barnhart, C., Jaillet, P., 2018b. The Stochastic Container Relocation Problem. Transp. Sci. 52, 1035–1058.

Giuliano, G. and O'Brien, T., 2007. Reducing port-related truck emissions: The terminal gate appointment system at the Ports of Los Angeles and Long Beach. Transp. Res. Part D Transp. Environ. 12(7), 460-473.

Gharehgozli, A.H., Roy, D. and de Koster, R., 2016. Sea container terminals: New technologies and OR models. Marit. Econ. Logist. 18(2),103-140.

Huynh, N., Walton, C.M. and Davis, J., 2004. Finding the number of yard cranes needed to achieve desired truck turn time at marine container terminals. Transp Res Record. 1873(1), 99-108.

Huynh, N., Zumerchik, J., 2010. Analysis of Stacking Priority Rules to Improve Drayage Operations Using Existing and Emerging Technologies. Transp. Res. Rec. 2162, 1–8.

Jovanovic, R. and Voß, S., 2014. A chain heuristic for the blocks relocation problem. Comput. Ind. Eng. 75, 79-86.

Jin, B., Zhu, W. and Lim, A., 2015. Solving the container relocation problem by an improved greedy look-ahead heuristic. Eur. J. Oper. Res. 240(3), 837-847.

Kim, K.H., Hong, G.P., 2006. A heuristic rule for relocating blocks. Comput. Oper. Res. 33, 940–954.

Kim, K.H., Park, Y.M., Ryu, K.R., 2000. Deriving decision rules to locate export containers in container yards. Eur. J. Oper. Res. 124(1), 89-101.

Ku, D., Arthanari, T.S., 2016a. Container relocation problem with time windows for container departure. Eur. J. Oper. Res. 252, 1031–1039.

Ku, D., Arthanari, T.S., 2016b. On the abstraction method for the container relocation problem. Comput. Oper. Res. 68, 110–122.

Lee, Y., & Lee, Y. J., 2010. A heuristic for retrieving containers from a yard. Comput. Oper. Res, 37(6), 1139-1147.

Lee, C.Y., Song, D.P., 2017. Ocean container transport in global supply chains: Overview and research opportunities. Transp. Res. Part B Methodol. 95, 442–474.

Lehnfeld, J., Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. Eur. J. Oper. Res. 239, 297–312.

Lin, D.Y., Lee, Y.J. and Lee, Y., 2015. The container retrieval problem with respect to relocation. Transp. Res. Part C Emerg. Technol. 52.132-143.

Li, N., Chen, G., Ng, M., Talley, W.K. and Jin, Z., 2019. Optimized appointment scheduling for export container deliveries at marine terminals. Marit Policy Manag. 1-23.

López-Plata, I., Expósito-Izquierdo, C., Lalla-Ruiz, E., Melián-Batista, B., Moreno-Vega, J.M., 2017. Minimizing the Waiting Times of block retrieval operations in stacking facilities. Comput. Ind. Eng. 103, 70–84.

Mongelluzzo, B. 2016. Long Beach automated terminal expects fastest harbor truck turns, JOC.com, Feb 11, 2016. https://www.joc.com/port-news/us-ports/port-long-beach/long-beach-automated-terminal-expects-fastest-harbor-truck-turns_20160211.html (accessed. 21/02/2020).

Mongelluzzo, B. 2019. LA-LB truckers: We need true interoperable chassis pools, JOC.com, Oct 29 2019. https://www.joc.com/port-news/la-lb-truckers-we-need-true-interoperable-chassis-pools_20191029.html. (accessed. 29/02/2020).

Mongelluzzo, B. 2020. Technology, mandatory truck slots push LA-LB turn times to near six-year low, JOC.com, Jan 13 2020.https://www.joc.com/port-news/terminal-operators/technology-mandatory-truck-slots-push-la-lb-turn-times-almost-six-year-low_20200113.html. (accessed. 22/02/2020).

Patrick. http://www.patrick.com.au/documents/VBS-Charges-Melbourne-July-2018.pdf (accessed. 28/02/2020).

Petering, M.E.H., Hussein, M.I., 2013. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. Eur. J. Oper. Res. 231, 120–130.

Pham, Q., Huynh, N., Xie, Y., 2011. Estimating Truck Queuing Time at Marine Terminal Gates. Transp. Res. Rec. J. Transp. Res. Board 2222, 43–53.

Port Botany. https://www.adventintermodal.com/customers/port-botany-new-south-wales-australia/ (accessed. 29/02/2020).

Port Metro Vancouver. https://cleanairactionplan.org/documents/final-2017-clean-air-action-plan-update.pdf/ (accessed. 21/02/2020).

Quispe, K.E.Y., Lintzmayer, C.N. and Xavier, E.C., 2018. An exact algorithm for the blocks relocation problem with new lower bounds. Comput. Oper. Res. 99, 206-217.

Russell, S.J. and Norvig, P., 2016. Artificial intelligence: a modern approach. Third Ed. Pearson Education Limited, Malaysia, pp. 80.

Stahlbock, R., Voß, S., 2008. Operations research at container terminals: A literature update. OR Spectr. 30, 1–52.

Saanen, Y. A., 2011. Modeling techniques in planning of terminals: The quantitative approach. In Böse, J.W. (Ed.), Handbook of Terminal Planning. Operations Research/Computer Science Interfaces Series. Springer, New York. pp. 83–102.

Talley, W.K. and Ng, M., 2016. Port multi-service congestion. Transp. Res. Part E Logist. Transp. Rev. 94, 66-70.

Tanaka, S., Takii, K., 2016. A faster branch-and-bound algorithm for the block relocation problem. IEEE Trans. Autom. Sci. Eng. 13, 181–190.

Tang, L., Jiang, W., Liu, J., Dong, Y., 2015. Research into container reshuffling and stacking problems in container terminal yards. IIE Trans. 47, 751–766.

Ting, C.J. and Wu, K.C., 2017. Optimizing container relocation operations at container yards with beam search. Transp. Res. Part E Logist. Transp. Rev. 103, 17-31.

Tong, X., Woo, Y.J., Jang, D.-W., Kim, K.H., 2015. Heuristic Rules Based on a Probabilistic Model and a Genetic Algorithm for Relocating Inbound Containers with Uncertain Pickup Times. Int. J. Ind. Eng. Appl. Pract. 22, 93–101.

Ünlüyurt, T., Aydın, C., 2012. Improved rehandling strategies for the container retrieval process. J. Adv. Transp. 46, 378–393.

Wan, Y., Liu, J., Tsai, P.-C., 2009. The Assignment of Storage Locations to Containers for a Container Stack. Nav. Res. Logist. 56, 699–713.

Yang, M., Allen, T.T., Fry, M.J., Kelton, W.D., 2013. The call for equity: simulation optimization models to minimize the range of waiting times. IIE Trans. 45, 781–795.

Zehendner, E., Caserta, M., Feillet, D., Schwarze, S., Voß, S., 2015. An improved mathematical formulation for the blocks relocation problem. Eur. J. Oper. Res. 245(2), 415–422.

Zehendner, E., Feillet, D., 2014. A branch and price approach for the container relocation problem. Int. J. Prod. Res. 52, 7159–7176.

Zehendner, E., Feillet, D., Jaillet, P., 2017. An algorithm with performance guarantee for the Online Container Relocation Problem. Eur. J. Oper. Res. 259, 48–62.

Zeng, Q., Feng, Y., Yang, Z., 2019. Integrated optimization of pickup sequence and container rehandling based on partial truck arrival information. Comput. Ind. Eng. 127, 366–382.

Zhang, C., Chen, W., Shi, L., Zheng, L., 2010. A note on deriving decision rules to locate export containers in container yards. Eur. J. Oper. Res. 205, 483–485.

Zhao, W., Goodchild, A. V., 2010. The impact of truck arrival information on container terminal rehandling. Transp. Res. Part E Logist. Transp. Rev. 46, 327–343.

Zhen, L., Jiang, X., Lee, L.H., Chew, E.P., 2013. A Review on Yard Management in Container Terminals. Ind. Eng. Manag. Syst. 12, 289–304.

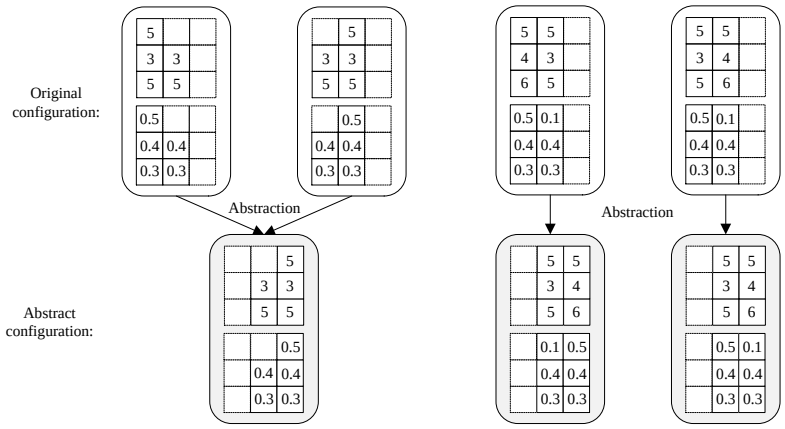
Zhu, W., Qin, H., Lim, A., Zhang, H., 2012. Iterative deepening A* algorithms for the container relocation problem. IEEE Trans. Autom. Sci. Eng. 9, 710–722.

Zweers, B.G., Bhulai, S. and van der Mei, R.D., 2020. Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem. Eur. J. Oper. Res. 283(3), 954–971.

Appendix A. Some illustrations of the exact algorithms

A.1. Illustration of the abstraction technique

Fig. A.1(a) shows the application of the abstract technique on the node 10 and node 11 in Fig. 4. By $Abstract(10)$ and $Abstract(11)$, node 10 and node 11 are projected to the same abstract configuration. This means if $f(10)$ is known, $f(11)$ can be directly returned to be $f(10)$ without further branching. Fig. A.1(b) illustrates two unequivalent abstract configurations due to the difference in their abstract preference configurations although they have the same abstract priority configuration.



(a) Equivalent configuration (b) Unequivalent configuration

Fig. A.1. Illustration of the abstraction technique

A.2. A sample decision tree developed by the extended APBFS algorithm

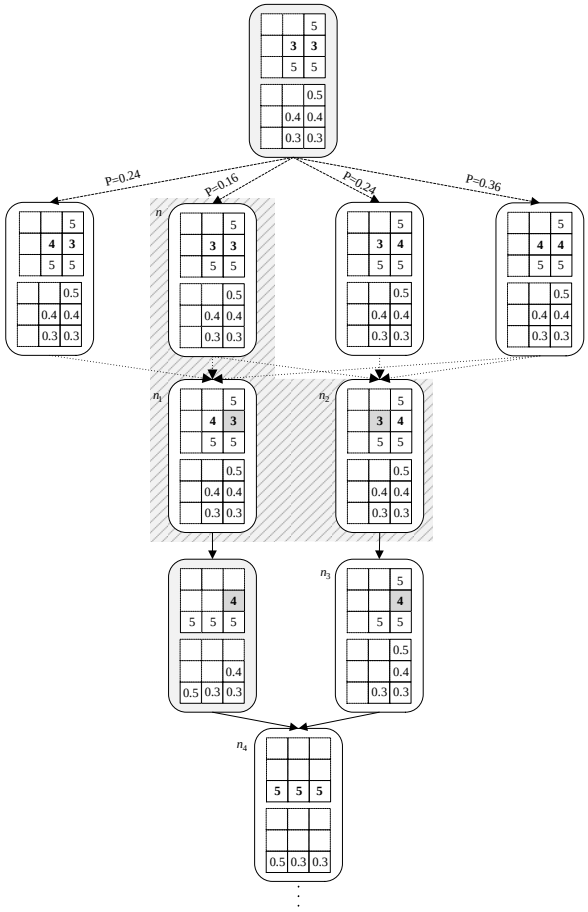


Fig. A.2. A sample decision tree developed by the extended APBFS algorithm.

Fig. A.2 presents a decision tree developed by the extended APBFS algorithm. The initial node in this decision tree is the abstract configuration of the last node in the decision tree of Fig. 4. Let us focus on the nodes highlighted with upward diagonal background: n, n_1, and n_2, to illustrate the consideration of the secondary objective. n is a SD node with $\Phi_n^{APBFS}=\{n_1,n_2\}$, and $f(n)$=1. As $f(n_1)=f(n_2)=1=f(n)$, we calculate the waiting time indicator of n_1 and n_2 by Algorithm 2. We obtain that $w(n_2)=1<w(n_1)=2$. Therefore, the best offspring of node n is n_2 (step 4.7-4.8 in Algorithm 3). Note that in the APBFS algorithm, if n_1 is first added into Φ_n^{APBFS}, n_2 will not be able to be included into Φ_n^{APBFS} as $lb(n_2)=1$ is not less than $f(n_1)$, which means that we lose the opportunity to find the optimal solution with regard to the secondary objective.

Appendix B. Details of the heuristics

B.1. Calculating the BI and DI of the EM extension heuristic

Fig. B.1 is used for illustration, which shows how the EM extension heuristic makes decisions on a simple example where the truck arrival sequence of the three containers in the first batch (u_7,u_{10},u_5) has been revealed. The container in the shaded slot is the target container to be retrieved. The container in the upward diagonal slot is the blocking container to be relocated at the current step. The numbers under the priority matrix correspond to the $m(s)$ of each candidate stack, and the value of M is also given.

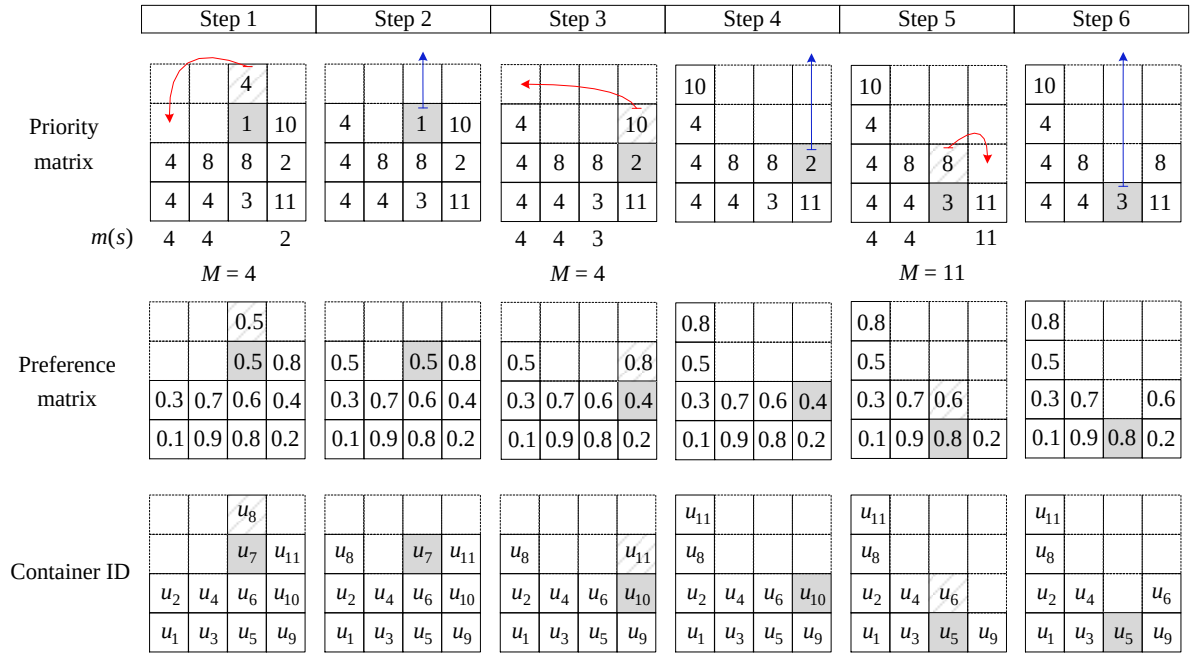


Fig. B.1. Decisions by the EM extension heuristic on an example

(1) Method of computing BI

If $M=l_c$, EM extension selects the stack with the minimum $BI(s')$ to minimize the probability of c being blocking if c is relocated to stack s'. Given a configuration B with M, a stack s where c is located before being relocated, a stack s' that satisfies $m(s')=M$, $BI(s')$ is computed as follows. Step 1 in Fig. B.1 is used to illustrate the computing method. Without special instruction, the 'stack 1' used in this sub-section refers to the stack 1 in the configuration under step 1 in Fig. B.1.

Let $M_{s'}=\{c_1,...,c_N\}$, $N=|M_{s'}|$, be the set of containers labeled M and located in s'. We first compute the probability that c is not blocking if relocated to s', i.e., $1-BI(s')$. Let us consider the two cases in terms of the sub-batch of c: #1) c is in the former sub-batch; #2) c is in the latter sub-batch.

#1. c is in the former sub-batch.

Under case 1, we consider two mutually exclusive sub-cases (#1.1 and #1.2) in terms of the sub-batch of containers in

$M_{s'}$.

#1.1. At least one container $c_i \in M_{s'}$ is in the former sub-batch.

There are totally $\sum_{k=1}^{N}\binom{N}{k}$ scenarios that satisfy #1.1. Let $Comb_k = \left\{comb_k^1, comb_k^2, ..., comb_k^{\binom{N}{k}}\right\}$ denote the set of all scenarios represented by the combinations of the *N* elements in $M_{s'}$ taken *k*, $k = \{1,...,N\}$. The size of $Comb_k$ is $\binom{N}{k}$. Each element $comb_k^i \in Comb_k$, $i \in \left\{1,...,\binom{N}{k}\right\}$, represents a scenario where the elements in $comb_k^i$ are in the former sub-batch. For example (see Fig. B.1), in the configuration of step 1, $M_1 = \{u_1, u_2\}$, *N*=2, $Comb_1 = \left\{comb_1^1, comb_1^2\right\}$, wherein $comb_1^1 = \{u_1\}$, $comb_1^2 = \{u_2\}$, and $Comb_2 = \{comb_2^1\}$, wherein $comb_2^1 = \{u_1, u_2\}$. The probability of $Comb_k$ is equal to $\sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})$. Then the probability that *c* is not blocking in the scenario set $Comb_k$ is equal to $\sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1)\cdot p_{c_i}$. Considering all combinations from *k*=1 to *k*=*N*, we have the probability that *c* is not blocking in case 1.1, which is equal to $\sum_{k=1}^{N}\sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1)\cdot p_{c_i}$. Taking stack 1 for example, we have

$$\sum_{k=1}^{N}\sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1)\cdot p_{c_i} = \sum_{k=1}^{2}\sum_{i=1}^{\binom{2}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1)\cdot p_{c_i} = (0.3\times 0.9/2 + 0.1\times 0.7/2 + 0.3\times 0.1/3)\times 0.5 = 0.09.$$

#1.2. All containers in $M_{s'}$ are in the latter sub-batch.

In this case, *c* is surely not blocking. Then the probability that *c* is not blocking in case 1.2 is equal to $\prod_{i=1}^{N}(1-p_{c_i})\cdot p_c$. Taking stack 1 for example, we have $\prod_{i=1}^{N}(1-p_{c_i})\cdot p_c = \prod_{i=1}^{2}(1-p_{c_i})\cdot p_c = 0.7\times 0.9\times 0.5 = 0.315$.

#2. *c* is in the latter sub-batch.

In this case, there exists only one scenario in which it is possible that *c* is not blocking, that is, all containers in $M_{s'}$ are in the latter sub-slot. Then the probability that *c* is not blocking in case 2 is equal to $\prod_{i=1}^{N}(1-p_{c_i})\cdot(1-p_c)/(N+1)$. Taking stack 1 for example, we have $\prod_{i=1}^{N}(1-p_{c_i})\cdot(1-p_c)/(N+1) = \prod_{i=1}^{2}(1-p_{c_i})\cdot(1-p_c)/3 = 0.7\times 0.9\times 0.5/3 = 0.105$.

The above cases exhaust all the possible scenarios of the sub-batches of the containers labeled *M*. Therefore, by summing the above expressions, we have the probability that *c* is not blocking if relocated to s', i.e., $1-BI(s')$, as expressed in Eq. (A.1):

$$1-BI(s') = \sum_{k=1}^{N}\sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1)\cdot p_{c_i} + \prod_{i=1}^{N}(1-p_{c_i})\cdot p_c + \prod_{i=1}^{N}(1-p_{c_i})\cdot(1-p_c)/(N+1)$$

$$= \sum_{k=1}^{N} \sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1) \cdot p_{c_i} + \prod_{i=1}^{N} (1-p_{c_i}) \cdot \left(p_c + (1-p_c)/(N+1)\right) \quad \text{(A.1)}$$

Finally, we have the probability of c being blocking if relocated to s', i.e., $BI(s')$, as calculated by Eq. (A.2).

$$BI(s') = 1 - \sum_{k=1}^{N} \sum_{i=1}^{\binom{N}{k}} \prod_{c_j \in comb_k^i} p_{c_j} \prod_{c_j \notin comb_k^i} (1-p_{c_j})/(k+1) \cdot p_{c_i} - \prod_{i=1}^{N} (1-p_{c_i}) \cdot \left(p_c + (1-p_c)/(N+1)\right) \quad \text{(A.2)}$$

For example (see Fig. B.1), in the configuration of step 1, the BI of stack 1 is calculated as $BI(1) = 1 - (0.09 + 0.315 + 0.105) = 0.49$. Also, we obtain $BI(2)$=0.7 by Eq. (A.2). As $BI(1)<BI(2)$, stack 1 is selected as the relocating stack at step 1 by the EM extension heuristic.

(2) Method of computing DI

If $M < l_c$, which means that it is unavoidable that c will be relocated again in the future, EM extension selects the stack with the minimum $DI(s')$ to delay the next relocation of c. Given a configuration B with M, a stack s where c is located before being relocated, a stack s' that satisfies $m(s') = M$, $DI(s')$ is computed as follows. Step 3 in Fig. B.1. is used to illustrate the computing method. Without special instruction, the 'configuration' used in this sub-section refers to the configuration at step 3 in Fig. B.1.

Let $M_B = \{c_1, ..., c_{L+1}\}$, $|M_B| = L+1$, be the set of all containers labeled M in configuration B, and $M_{s'}$ be the set of containers labeled M and located in s'. For example, in the illustrated configuration, $M_B = \{u_1, u_2, u_8, u_3\}$, L=3, $M_1 = \{u_1, u_2, u_8\}$, and $M_2 = \{u_3\}$. Given a candidate stack s', we first compute the probability of each container $c_i \in M_{s'}$ being the first one to be retrieved among the containers in M_B, denoted by $DI'(c_i)$. Since the retrieval of any container $c_i \in M_{s'}$ will cause the next relocation of c if c is relocated to s', by definition, we have $DI(s') = \sum_{c_i \in M_{s'}} DI'(c_i)$.

Now let us consider a container $c_i \in M_{s'}$ and compute $DI'(c_i)$. Suppose all the containers in $M_B \setminus c_i$ are located in a dummy stack and c_i is the container to be relocated to this stack. Then $DI'(c_i)$ is equal to the probability that c_i is not blocking if relocated to this dummy stack. Therefore, using the Eq. (A.1) of calculating $1 - BI(s')$, $DI'(c_i)$ is computed by Eq. (A.3).

$$DI'(c_i) = \sum_{k=1}^{L} \sum_{n=1}^{\binom{L}{k}} \prod_{c_j \in cmb_{c_i,k}^n} p_{c_j} \cdot \prod_{c_j \notin cmb_{c_i,k}^n} (1-p_{c_j})/(k+1) \cdot p_{c_i} + \prod_{c_j \in M_B \setminus c_i} (1-p_{c_j}) \cdot p_{c_i} + \prod_{c_j \in M_B \setminus c_i} (1-p_{c_j}) \cdot (1-p_{c_i})/(L+1)$$

$$= \sum_{k=1}^{L} \sum_{n=1}^{\binom{L}{k}} \prod_{c_j \in cmb_{c_i,k}^n} p_{c_j} \cdot \prod_{c_j \notin cmb_{c_i,k}^n} (1-p_{c_j})/(k+1) \cdot p_{c_i} + \prod_{c_j \in M_B \setminus c_i} (1-p_{c_j}) \cdot \left(p_{c_i} + (1-p_{c_i})/(L+1)\right) \quad \text{(A.3)}$$

wherein, $cmb_{c_i,k}^n \in Cmb_{c_i,k}$. $Cmb_{c_i,k} = \left\{cmb_{c_i,k}^1, cmb_{c_i,k}^2, ..., cmb_{c_i,k}^{\binom{L}{k}}\right\}$ denotes the set of all scenarios represented by the combinations of the L elements in $M_B \setminus c_i$ ($c_i \in M_{s'}$) taken k, $k = \{1, ..., L\}$. The size of $Cmb_{c_i,k}$ is $\binom{L}{k}$.

Therefore, $DI(s') = \sum_{c_i \in M_{s'}} DI'(c_i)$ is computed by Eq. (A.4).

$$DI(s') = \sum_{c_i \in M_{s'}} \left[\sum_{k=1}^{L} \sum_{n=1}^{\binom{L}{k}} \prod_{c_j \in cmb^n_{c_i,k}} p_{c_j} \cdot \prod_{c_j \notin cmb^n_{c_i,k}} (1-p_{c_j})/(k+1) \cdot p_{c_i} + \prod_{c_j \in M_B \setminus c_i} (1-p_{c_j}) \cdot \left(p_{c_i} + (1-p_{c_i})/(L+1) \right) \right] \quad \text{(A.4)}$$

Let us calculate $DI'(u_8)$ in the illustrated configuration. $Cmb_{u_8,1} = \left\{ cmb^1_{u_8,1}, cmb^2_{u_8,1}, cmb^3_{u_8,1} \right\}$, wherein $cmb^1_{u_8,1} = \{u_1\}$, $cmb^2_{u_8,1} = \{u_2\}$, and $cmb^3_{u_8,1} = \{u_3\}$; $Cmb_{u_8,2} = \left\{ cmb^1_{u_8,2}, cmb^2_{u_8,2}, cmb^3_{u_8,2} \right\}$, wherein $cmb^1_{u_8,2} = \{u_1, u_2\}$, $cmb^2_{u_8,2} = \{u_1, u_3\}$, and $cmb^3_{u_8,2} = \{u_2, u_3\}$; $Cmb_{u_8,3} = \left\{ cmb^1_{u_8,3} \right\}$, wherein $cmb^1_{u_8,3} = \{u_1, u_2, u_3\}$. For $k = \{1,...,3\}$, we calculate the first term of Eq. (A.3):

For *k*=1, $\sum_{n=1}^{\binom{3}{1}} \prod_{c_j \in cmb^n_{u_8,1}} p_{c_j} \cdot \prod_{c_j \notin cmb^n_{u_8,1}} (1-p_{c_j})/(1+1) \cdot p_{u_8} = (0.1 \times 0.7 \times 0.1 + 0.3 \times 0.9 \times 0.1 + 0.9 \times 0.7 \times 0.9)/2 \times 0.5 = 0.15025$

For *k*=2, $\sum_{n=1}^{\binom{3}{2}} \prod_{c_j \in cmb^n_{u_8,2}} p_{c_j} \cdot \prod_{c_j \notin cmb^n_{u_8,2}} (1-p_{c_j})/(2+1) \cdot p_{u_8} = (0.1 \times 0.3 \times 0.1 + 0.1 \times 0.9 \times 0.7 + 0.3 \times 0.9 \times 0.9)/3 \times 0.5 = 0.0515$

For *k*=3, $\sum_{n=1}^{\binom{3}{3}} \prod_{c_j \in cmb^n_{u_8,3}} p_{c_j} \cdot \prod_{c_j \notin cmb^n_{u_8,3}} (1-p_{c_j})/(3+1) \cdot p_{u_8} = 0.1 \times 0.3 \times 0.9/4 \times 0.5 = 0.003375$

Summing the above expressions, we have,

$$\sum_{k=1}^{3} \sum_{n=1}^{\binom{3}{k}} \prod_{c_j \in cmb^n_{u_8,k}} p_{c_j} \cdot \prod_{c_j \notin cmb^n_{u_8,k}} (1-p_{c_j})/(k+1) \cdot p_{u_8} = 0.15025 + 0.0515 + 0.003375 = 0.205125$$

Then, we calculate the second term of Eq. (A.3):

$$\prod_{c_j \in M_B \setminus u_8} (1-p_{c_j}) \cdot \left(p_{u_8} + (1-p_{u_8})/(L+1) \right) = (0.9 \times 0.7 \times 0.1) \times (0.5 + 0.5/4) = 0.039375$$

By Eq. (A.4), we have $DI'(u_8) = 0.205125 + 0.039375 = 0.2445$. In the same way, we obtain $DI'(u_2) = 0.1385$, $DI'(u_1) = 0.0485$, and $DI'(u_3) = 0.5685$. Therefore, $DI(1) = DI'(u_8) + DI'(u_2) + DI'(u_1)$ =0.2445+0.1385+0.0485=0.4315, and $DI(2) = DI'(u_3) = 0.5685$. As $DI(1) < DI(2)$, stack 1 is selected as the target relocating stack at step 3 by the EM extension heuristic.

B.2. An illustrative example for the SEM heuristic

We use Fig. B.2. to illustrate the sequencing decision of the SEM heuristic. In the initial configuration X_0, there are five batches of containers. The truck arrival information of the first batch is revealed to be that u_7 is in the first sub-batch and u_5 and u_{10} are in the second sub-batch, as shown in bold in Step 1. Now we present the decisions to retrieve the first batch containers. The container in the shaded slot represents the target container to be retrieved. The container in the upward diagonal slot represents the blocking container to be relocated. At Step 1, $lmin = 1$, $\Theta = \{u_7\}$, and thus there is no doubt that u_7 is selected as the target container. After retrieving u_7, $lmin = 2$, $\Theta = \{u_5, u_{10}\}$, and $r(u_5) = r(u_{10}) = 1$. As $r(u_5) = r(u_{10})$, the SEM selects u_{10} as the target container arbitrarily. After Step 4, $lmin = 3$, $\Theta = \{u_5\}$, and thus u_5 is selected as the target container out of question.

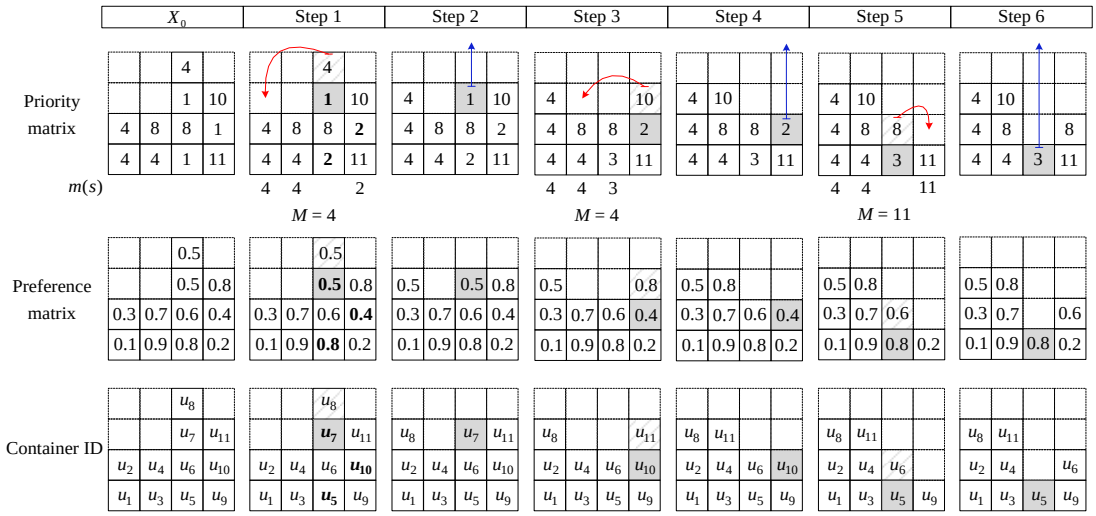


Fig. B.2. Decisions by the SEM heuristic on an example

B.3. Calculating the BIS and DIS of the SEM heuristic

Fig. B.2 is used for illustration.

(1) Method of computing BIS

Given a configuration *B* with *M*, a stack *s* where *c* is located before being relocated, a stack s' that satisfies $m(s')=M$, the method of computing $BIS(s')$ is introduced here. Let $M_{s'}=\{c_1,...,c_N\}$, $N=|M_{s'}|$, be the set of containers labeled *M* and located in s'. Container *c* being blocking if relocated to s' occurs only in the scenario where *c* is in the latter sub-batch and there is at least one container $c_i \in M_{s'}$ in the former sub-batch. Therefore, we have,

$$BIS(s')=(1-p_c)\left(1-\prod_{c_i\in M(s')}(1-p_{c_i})\right) \tag{A.5}$$

The term $\prod_{c_i\in M(s')}(1-p_{c_i})$ of Eq. (A.5) is the probability that all the containers in $M_{s'}$ are in the latter sub-batch, and thus $\left(1-\prod_{c_i\in M(s')}(1-p_{c_i})\right)$ is the probability that at least one of them is in the former sub-batch.

Take the configuration at step 1 (see Fig. B.2) for example. $c=u_8$, *M*=4, and thus stack 1 and stack 2 are candidate stacks. By Eq. (A.5), $BIS(1)=(1-0.5)(1-0.7\times0.9)=0.185$, $BIS(2)=(1-0.5)(1-0.1)=0.45$. As *BIS*(1)<*BIS*(2), stack 1 is selected for relocating u_8.

(2) Method of computing DIS

In the sequencing rule introduced above, the one with the lowest number of blocking containers among the containers with the smallest label is selected as the target container, ties being broken arbitrarily. Therefore, a container c_i is surely being the first one to be retrieved in its batch only in the situation that satisfies the following two conditions: 1) c_i is in the former sub-batch; 2) c_i is with the lowest number of blocking containers (i.e., r_{c_i}) among the containers in the former sub-batch. The second condition means that all the containers above c_i labeled *M* must be in the latter sub-batch and for each stack s' that satisfies $m(s')=M$, if there are containers labeled *M* among the top $r_{c_i}+1$ number of containers (if any) in stack s', these containers must be in the latter sub-batch.

Given a configuration *B* with *M*, a stack *s* where *c* is located before being relocated, a stack $s'\in S_M$, wherein S_M is the set of stack that satisfies $m(s')=M$, the method of computing $DIS(s')$ is introduced here. Let $M_{s'}$ be the set of

containers labeled M located in s'. Given a candidate stack s', we first compute the probability of each container $c_i \in M_{s'}$ surely being the first one to be retrieved in its batch, denoted by $DIS'(c_i)$. By definition, we have $DIS(s') = \sum_{c_i \in M_{s'}} DIS'(c_i)$. Let $T(n,s)$ denote the set of top n number of containers labeled M in stack s. Then, we have $DIS'(c_i) = p_{c_i} \cdot \sum_{c_j \in T(r(c_i),s')} (1-p_{c_j}) \sum_{s'' \in S_M \setminus s'} \sum_{c_j \in T(r(c_i)+1,s'')} (1-p_{c_j})$. By summing the $DIS'(c_i)$ of all containers $c_i \in M_{s'}$, we get,

$$DIS(s') = \sum_{c_i \in M_{s'}} p_{c_i} \cdot \sum_{c_j \in T(r(c_i),s')} (1-p_{c_j}) \sum_{s'' \in S_M \setminus s'} \sum_{c_j \in T(r(c_i)+1,s'')} (1-p_{c_j}) \quad \text{(A.6)}$$

Taking the configuration at step 3 (see Fig. B.2) for example, where $c = u_{11}$, M=4, $S_M = \{1,2\}$, let us compute $DIS'(u_2)$. $r(u_2) = 1$, $T(1,1) = \{u_8\}$, $T(2,2) = \{u_3\}$, and thus $\sum_{c_j \in T(r(c_i),s')} (1-p_{c_j}) = \sum_{c_j \in T(1,1)} (1-p_{c_j}) = (1-p_{u_8}) = 0.5$, $\sum_{s'' \in S_M \setminus s'} \sum_{c_j \in T(r(c_i)+1,s'')} (1-p_{c_j}) = \sum_{s'' \in \{2\}} \sum_{c_j \in T(2,s'')} (1-p_{c_j}) = (1-p_{u_3}) = 0.1$. Consequently, $DIS'(u_2) = 0.3 \times 0.5 \times 0.1 = 0.015$. Similarly, we get $DIS'(u_8) = 0.5$, $DIS'(u_1) = 0.0035$, and $DIS'(u_3) = 0.315$. By Eq. (A.6), we have $DIS(1) = DIS'(u_2) + DIS'(u_8) + DIS'(u_1) = 0.5 + 0.015 + 0.0035 = 0.5185$ and $DIS(2) = DIS'(u_3) = 0.315$. As $DIS(2) < DIS(1)$, stack 2 is selected for relocating u_{11}.

B.4. An illustrative example for the SEML heuristic

We use Fig. B.3, which continues Step 2 of Fig. B.2, to illustrate the sequencing rule of the SEML heuristic. For brevity, we only present the priority matrix. After step 2, $lmin = 2$, and $\Theta = \{u_5, u_{10}\}$. Because $r(u_5) = r(u_{10}) = 1$, both u_5 and u_{10} are potential target containers (H=2). Hence, we evaluate the contribution of the two feasible retrieval sequences respectively: $u_5 \to u_{10}$ and $u_{10} \to u_5$, through step 3 to step 6. The final configurations after implementing the two sequences are given at step 7. It is obvious that the two final configurations have the same lower bound. And because both of the sequences cause two realized relocations, their contributions are the same. Therefore, we can choose one arbitrarily from $u_5 \to u_{10}$ and $u_{10} \to u_5$ as the determined retrieval sequence for u_5 and u_{10}.

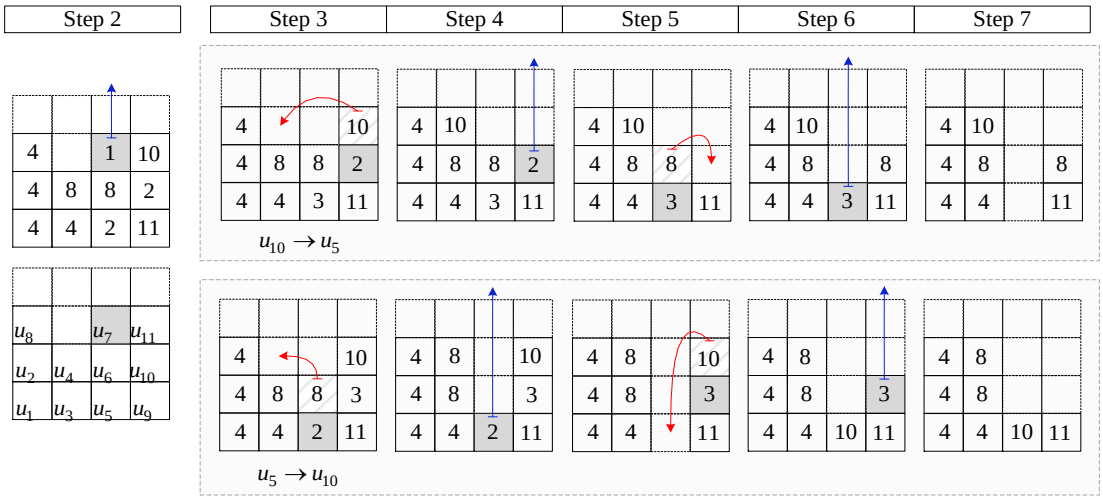


Fig. B.3. Decisions by the SEML heuristic on an example

Appendix C. Performance of the proposed models and exact algorithms

Note that in the tables, the "Opt" column represents the optimal expected number of relocations obtained by the corresponding exact algorithms; the "Rel" column represents the estimated number of relocations obtained by simulation.

C.1. Results of the extended APBFS algorithm, the SEML heuristic, and the lower bound

Table C.1 Performance of the extended APBFS algorithm, the SEML heuristics and lower bound for instances with small batches and 67% fill rate

				Extended APBFS						SEML		Gap	
T	*S*	*C*	lb	lb*	Opt	Solved	CPU(s)	Rel	AveWait (min)	Rel	AveWait (min)	Gap [Rel]	Gap [AveWait]
3	5	10	2.621	2.621	2.789	√	0.05	2.786	3.224	2.786	3.222	0.00%	-0.06%
	6	12	3.433	3.433	3.621	√	0.05	3.620	3.524	3.659	3.526	1.08%	0.06%
	7	14	3.708	3.708	3.756	√	0.06	3.759	3.506	3.786	3.506	0.72%	0.00%
	8	16	4.254	4.254	4.379	√	0.08	4.382	3.385	4.405	3.389	0.52%	0.12%
	9	18	4.592	4.592	4.815	√	0.11	4.816	3.408	4.846	3.408	0.62%	0.00%
	10	20	5.021	5.021	5.067	√	0.11	5.066	3.176	5.073	3.179	0.14%	0.09%
4	5	13	4.433	4.433	5.073	√	0.14	5.071	3.976	5.241	3.972	3.35%	-0.10%
	6	16	6.017	6.017	6.925	√	1.19	6.931	3.898	7.200	3.936	3.88%	0.97%
	7	19	6.042	6.042	6.927	√	1.16	6.929	3.465	7.026	3.475	1.40%	0.29%
	8	21	7.375	7.375	7.967	√	20.25	7.969	3.601	8.016	3.605	0.59%	0.11%
	9	24	8.775	8.775	9.259	√	11.80	9.257	3.619	9.353	3.619	1.04%	0.00%
	10	27	8.992	8.935	9.618	27	92.17	9.622	3.426	9.709	3.424	0.90%	-0.05%
5	5	17	7.358	7.058	8.802	26	24.26	8.802	4.145	9.187	4.172	4.37%	0.66%
	6	20	7.992	7.522	8.465	23	66.56	8.468	3.774	8.679	3.796	2.49%	0.59%
	7	23	9.475	9.091	10.339	22	340.50	10.340	3.934	10.569	3.945	2.21%	0.28%
	8	27	11.354	10.816	11.570	17	803.78	11.568	3.705	11.671	3.714	0.89%	0.24%
	9	30	12.879	12.388	13.436	10	728.19	13.436	3.669	13.555	3.673	0.89%	0.10%
	10	34	14.229	12.5	12.983	11	195.66	12.990	3.549	13.032	3.548	0.32%	-0.02%
6	5	20	9.496	8.958	11.183	15	613.80	11.181	4.085	12.006	4.148	7.38%	1.53%
	6	24	11.304	10.894	12.341	13	491.90	12.343	4.102	12.527	4.127	1.49%	0.60%
	7	28	13.258	13.75	15.375	2	253.16	15.357	3.864	15.359	3.918	0.01%	1.39%
	8	32	15.367	13.2	13.869	5	693.38	13.865	3.695	13.865	3.685	0.00%	-0.26%
	9	36	16.454	16.5	16.500	2	949.85	16.504	3.646	16.504	3.652	0.00%	0.18%
	10	40	19.375	15.25	15.250	1	232.22	15.259	3.857	15.259	3.880	0.00%	0.61%

Note: customer preference scenario: 50%

Table C.2 Performance of the extended APBFS algorithm, the SEML heuristics and lower bound for instances with large batches and 50% fill rate

				Extended APBFS						SEML		Gap	
T	*S*	*C*	lb	lb*	Opt	Solved	CPU(s)	Rel	AveWait (min)	Rel	AveWait (min)	Gap [Rel]	Gap [AveWait]
3	5	8	1.263	1.263	1.278	√	0.45	1.281	7.206	1.283	7.215	0.16%	0.12%
	6	9	1.317	1.317	1.334	√	0.32	1.336	7.388	1.336	7.389	0.00%	0.01%
	7	11	2.317	2.317	2.340	√	0.47	2.339	7.075	2.339	7.078	0.00%	0.04%
	8	12	1.971	1.971	1.973	√	0.55	1.969	6.499	1.969	6.499	0.00%	0.00%
	9	14	2.617	2.617	2.621	√	0.72	2.614	6.764	2.615	6.764	0.04%	0.00%
	10	15	2.908	2.908	2.914	√	0.86	2.915	6.708	2.915	6.710	0.00%	0.03%
4	5	10	2.546	2.546	2.703	√	0.56	2.702	6.967	2.779	6.991	2.85%	0.34%
	6	12	3.183	3.183	3.315	√	1.13	3.312	7.122	3.357	7.140	1.36%	0.25%
	7	14	3.550	3.550	3.645	√	4.50	3.648	7.473	3.662	7.479	0.38%	0.08%
	8	16	4.058	4.058	4.094	√	6.46	4.090	7.502	4.102	7.507	0.29%	0.07%
	9	18	5.233	5.233	5.274	√	19.65	5.273	7.408	5.297	7.409	0.46%	0.01%
	10	20	5.746	5.746	5.799	√	16.80	5.803	6.863	5.844	6.869	0.71%	0.09%
5	5	13	3.952	3.952	4.470	√	32.02	4.466	8.154	4.633	8.187	3.74%	0.40%
	6	15	4.844	4.821	5.181	29	84.30	5.181	7.586	5.323	7.610	2.74%	0.31%
	7	18	5.704	5.594	5.809	28	75.49	5.809	8.007	5.843	7.988	0.58%	-0.24%
	8	20	6.813	6.690	6.971	27	102.75	6.974	7.315	7.055	7.327	1.16%	0.17%
	9	23	7.950	7.894	8.206	26	419.04	8.205	7.691	8.274	7.701	0.84%	0.12%
	10	25	8.671	8.65	8.875	25	183.02	8.870	7.441	8.933	7.446	0.71%	0.07%
6	5	15	5.717	5.714	6.599	28	197.67	6.602	8.074	6.857	8.134	3.86%	0.75%
	6	18	6.404	6.125	6.840	26	273.82	6.842	8.244	7.056	8.257	3.13%	0.16%
	7	21	8.140	7.545	7.975	14	271.02	7.976	8.125	8.074	8.159	1.23%	0.42%
	8	24	8.985	8.351	8.668	13	548.95	8.670	7.690	8.730	7.696	0.70%	0.08%
	9	27	9.913	9.366	9.758	14	589.12	9.763	7.122	9.878	7.173	1.18%	0.72%
	10	30	11.552	10.714	10.884	12	207.75	10.890	7.468	10.891	7.469	0.00%	0.02%

Note: customer preference scenario: 50%

Table C.3 Performance of the extended APBFS algorithm, the SEML heuristics and lower bound for instances with large batches and 67% fill rate

				Extended APBFS						SEML		Gap	
T	*S*	*C*	lb	lb*	Opt	Solved	CPU(s)	Rel	AveWait (min)	Rel	AveWait (min)	Gap [Rel]	Gap [AveWait]
3	5	10	2.296	2.296	2.404	√	0.90	2.403	6.945	2.415	6.953	0.50%	0.12%
	6	12	3.113	3.113	3.239	√	1.21	3.245	7.468	3.277	7.477	0.99%	0.12%
	7	14	3.463	3.463	3.520	√	3.58	3.518	7.720	3.527	7.723	0.26%	0.04%
	8	16	4.100	4.100	4.176	√	3.36	4.179	7.281	4.186	7.282	0.17%	0.01%
	9	18	4.346	4.346	4.483	√	4.05	4.485	7.349	4.541	7.365	1.25%	0.22%
	10	20	4.717	4.717	4.755	√	3.40	4.755	6.937	4.760	6.940	0.11%	0.04%
4	5	13	4.127	4.127	4.635	√	27.60	4.631	8.342	4.829	8.368	4.28%	0.31%
	6	16	5.629	5.594	6.169	28	39.35	6.167	8.091	6.398	8.142	3.74%	0.62%
	7	19	5.754	5.728	6.318	28	175.06	6.307	7.310	6.397	7.320	1.43%	0.14%
	8	21	6.921	6.726	7.025	26	73.15	7.027	7.580	7.171	7.591	2.04%	0.15%
	9	24	8.229	7.859	8.201	24	320.01	8.208	7.655	8.314	7.663	1.30%	0.10%
	10	27	8.888	8.429	8.897	21	441.13	8.898	7.233	8.994	7.246	1.08%	0.18%
5	5	17	7.006	6.679	7.664	22	498.41	7.677	8.612	8.178	8.684	6.52%	0.84%
	6	20	7.677	6.863	7.398	16	151.72	7.402	7.596	7.703	7.644	4.07%	0.64%
	7	23	9.208	7.889	8.566	9	443.50	8.566	7.681	8.755	7.723	2.20%	0.54%
	8	27	10.838	10.271	10.696	6	1091.25	10.678	7.467	10.871	7.477	1.81%	0.13%
	9	30	12.546	10.219	10.914	4	858.35	10.912	7.368	10.955	7.381	0.39%	0.17%
	10	34	13.925	10.979	11.264	6	549.90	11.249	7.162	11.264	7.177	0.13%	0.21%
6	5	20	9.183	8.075	9.527	5	1567	9.529	8.042	10.463	8.162	9.80%	1.49%
	6	24	11.067	9.844	10.531	2	1019.19	10.506	7.841	10.590	7.871	0.80%	0.38%
	7	28	12.967	-	-	0	-	-	-	-	-	-	-
	8	32	14.829	11.75	11.781	1	694.38	11.745	8.051	11.740	8.050	-0.04%	-0.01%
	9	36	16.196	-	-	0	-	-	-	-	-	-	-
	10	40	18.935	-	-	0	-	-	-	-	-	-	-

Note: customer preference scenario: 50%

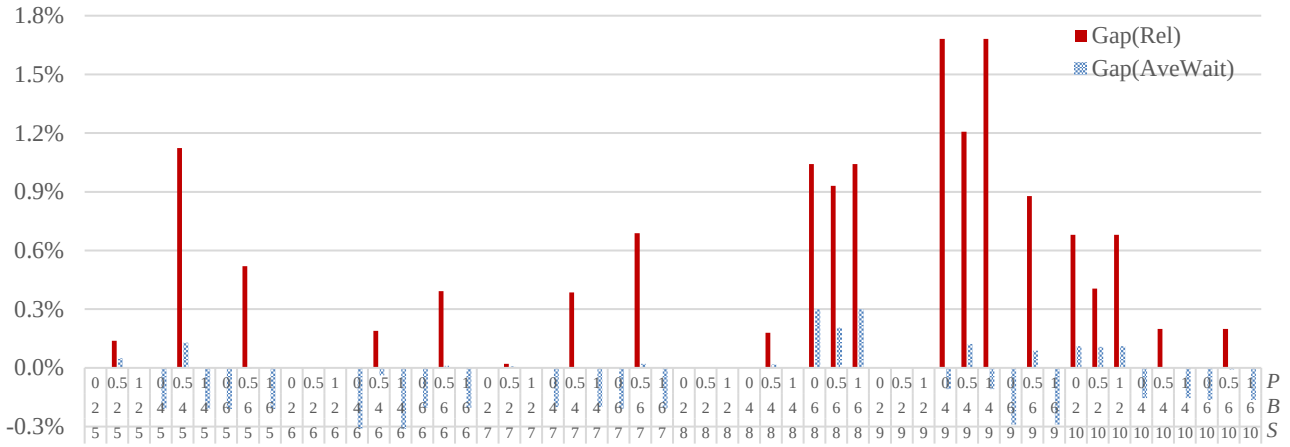
C.2. The calibrated results of Table 2 for the comparison between the Sooo model and the Sooo extension model

Table C.4. Comparison between the Sooo model and the Sooo extension model with small batches and 50% fill rate

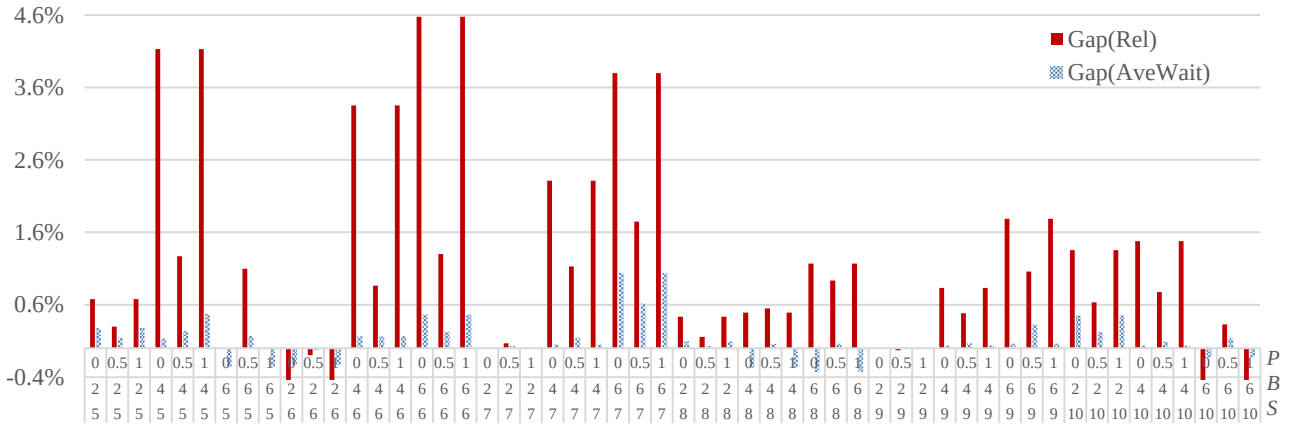
			Sooo- APBFS				Sooo extension- extended APBFS				Gap
T	*S*	*C*	Solved	CPU(s)	Rel	AveWait (min)	Solved	CPU(s)	Rel	AveWait (min)	[AveWait]
3	5	8	√	0.02	1.478	3.319	√	0.03	1.478	3.207	3.38%
	6	9	√	0.02	1.581	3.428	√	0.03	1.581	3.303	3.65%
	7	11	√	0.02	2.654	3.485	√	0.03	2.654	3.356	3.70%
	8	12	√	0.01	2.169	3.091	√	0.03	2.169	3.014	2.49%
	9	14	√	0.02	2.885	3.226	√	0.04	2.885	3.106	3.72%
	10	15	√	0.03	3.093	3.044	√	0.06	3.093	2.930	3.77%
4	5	10	√	0.02	2.855	3.534	√	0.03	2.855	3.411	3.49%
	6	12	√	0.03	3.466	3.532	√	0.05	3.466	3.351	5.12%
	7	14	√	0.04	3.941	3.485	√	0.05	3.941	3.324	4.61%
	8	16	√	0.16	4.559	3.556	√	0.20	4.559	3.390	4.67%
	9	18	√	0.29	5.523	3.691	√	0.34	5.523	3.474	5.87%
	10	20	√	0.72	6.015	3.334	√	0.80	6.015	3.181	4.59%
5	5	13	√	0.16	4.883	4.042	√	0.19	4.884	3.900	3.53%
	6	15	√	2.44	5.544	3.708	√	2.98	5.544	3.553	4.19%
	7	18	√	0.72	6.573	3.993	√	0.77	6.573	3.777	5.39%
	8	20	√	7.57	7.516	3.695	√	8.16	7.516	3.482	5.77%
	9	23	29	67.35	8.696	3.835	29	68.25	8.696	3.606	5.96%
	10	25	29	39.40	9.238	3.719	29	49.37	9.238	3.517	5.43%
6	5	15	√	4.56	6.999	4.034	√	5.06	6.999	3.886	3.66%
	6	18	√	5.48	7.728	4.162	√	6.18	7.728	3.959	4.89%
	7	21	23	*148.91*	*8.923*	*4.006*	22	160.87	8.923	3.787	5.46%
	8	24	22	150.07	9.881	3.998	22	164.07	9.882	3.748	6.24%
	9	27	18	100.24	10.538	3.661	18	104.47	10.538	3.419	6.60%
	10	30	19	*108.42*	*11.576*	*3.804*	18	108.52	11.576	3.580	5.90%

*Note: customer preference scenario: 50%

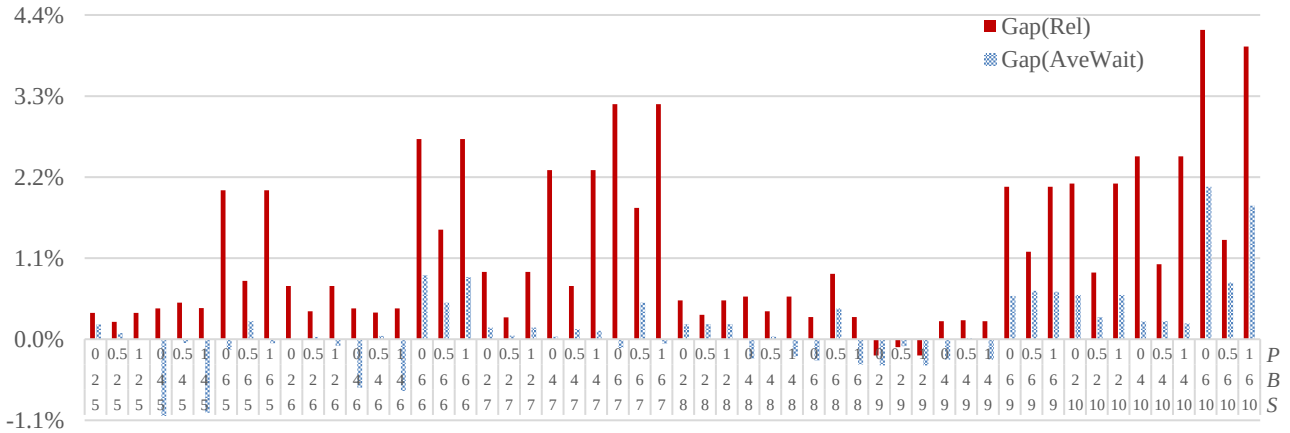
Appendix D. Comparisons between the SEM heuristic and the SEML heuristic



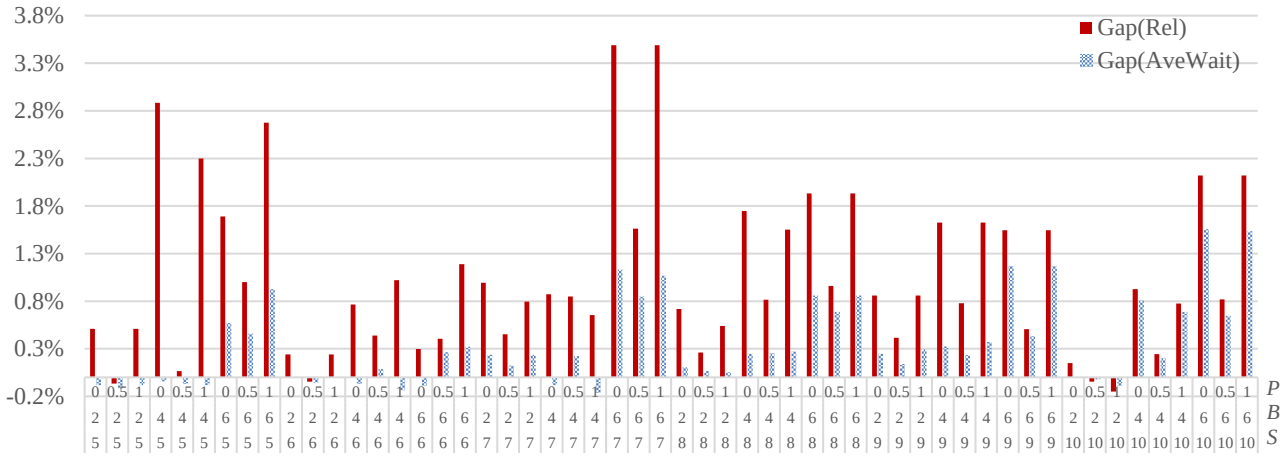
(a) Instances of three tiers



(b) Instances of four tiers



(c) Instances of five tiers



(d) Instances of six tiers

Fig. D.1. Comparisons between the SEM heuristic and the SEML heuristic for all the instances with 67% fill rate

Appendix E. Additional results for the effectiveness of the flexible service policy

E.1. Absolute reduction on the total number of relocations

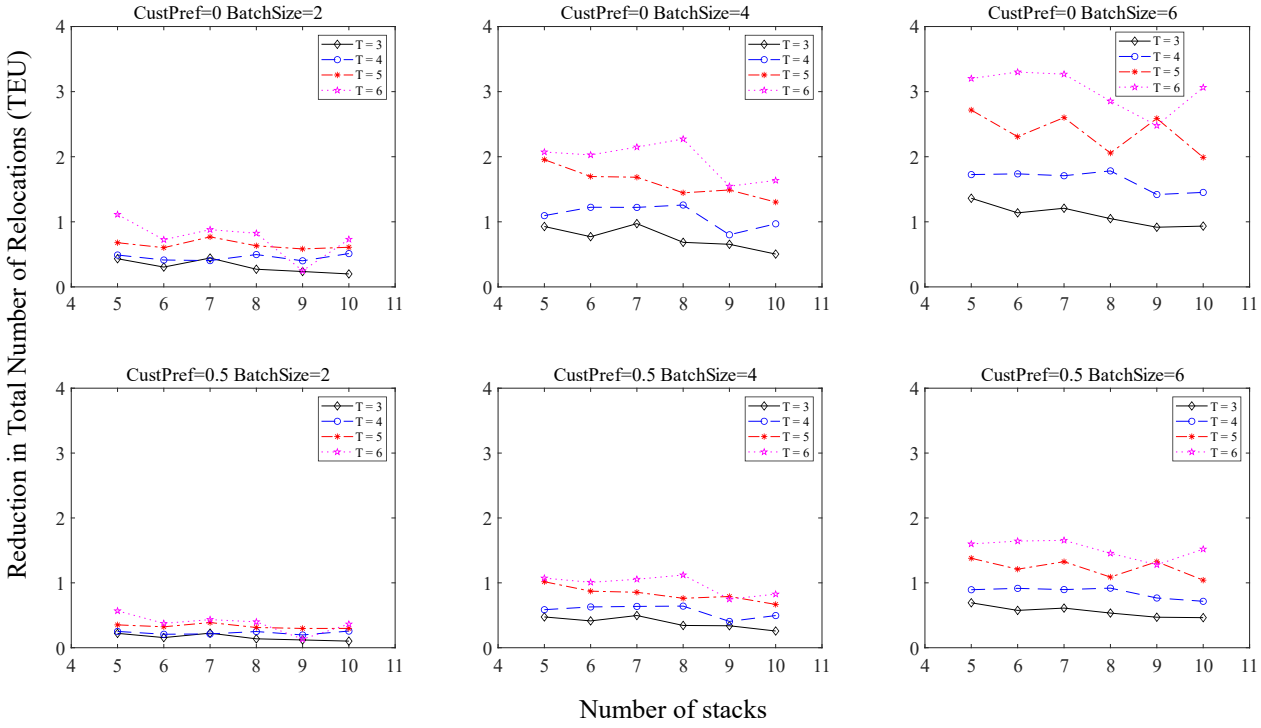


Fig. E.1 (a) Reduction on the total number of relocations by the flexible service policy for instances of 50% fill rate

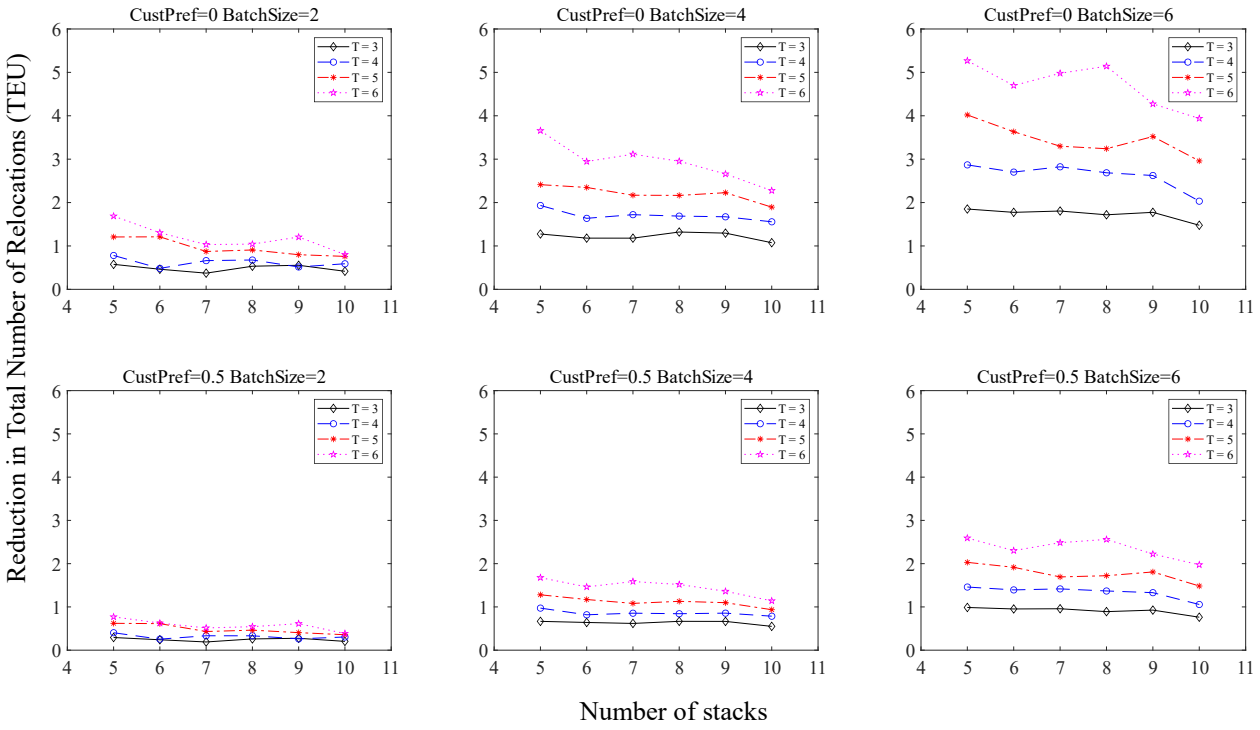


Fig. E.1 (b) Reduction on the total number of relocations by the flexible service policy for instances of 67% fill rate

E.2. Absolute reduction on the average relevant truck waiting time

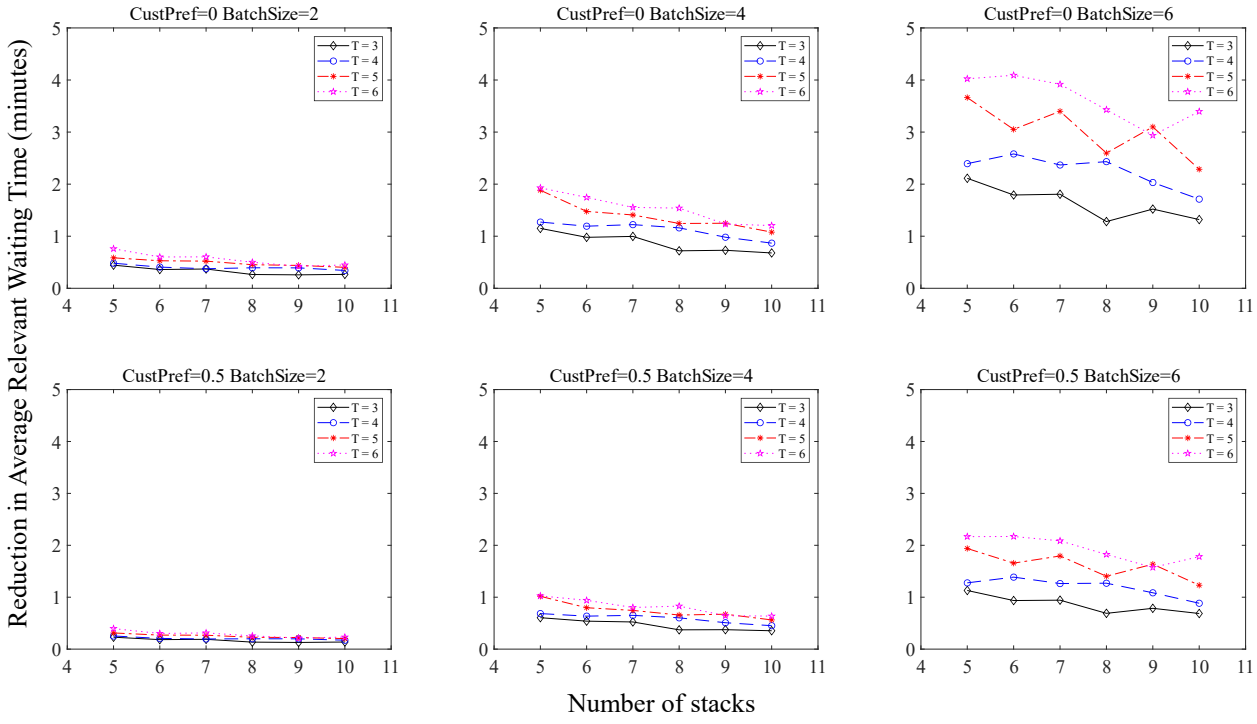


Fig. E.2 (a) Reduction on the average relevant waiting time by the flexible service policy for instances of 50% fill rate

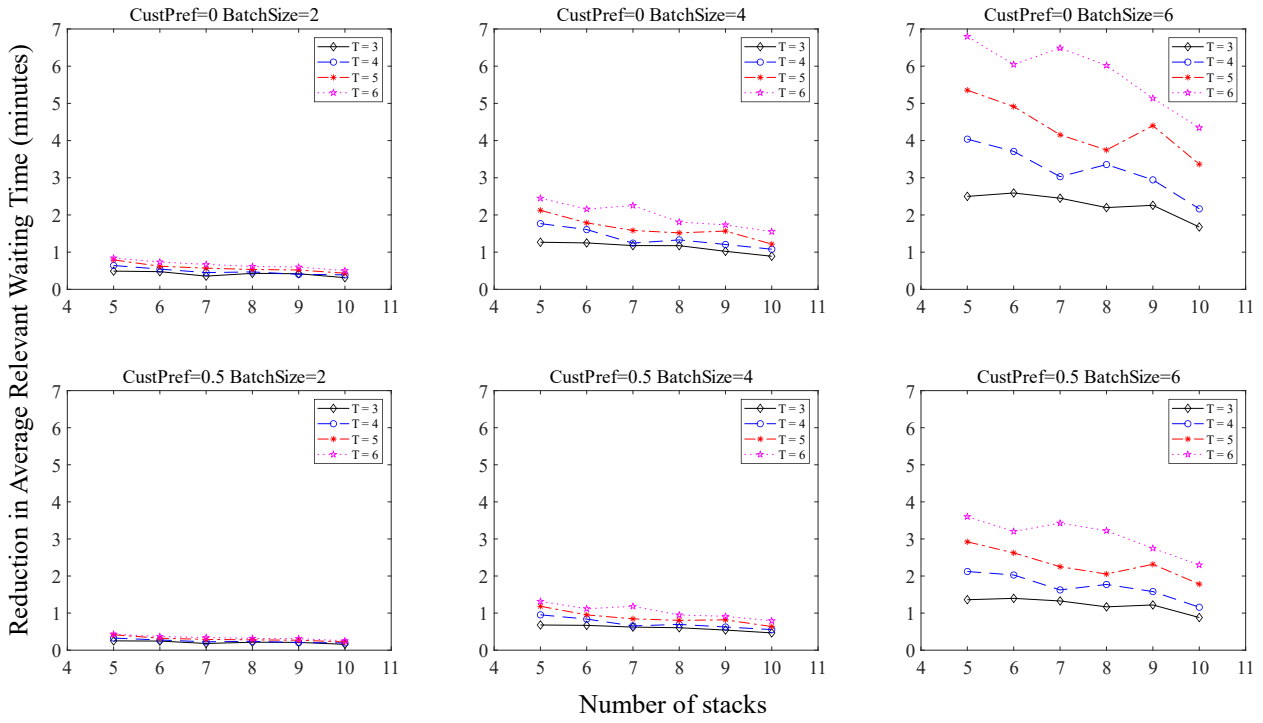


Fig. E.2 (b) Reduction on the average relevant waiting time by the flexible service policy for instances of 67% fill rate

E.3. Relative reduction on the average relevant truck waiting time

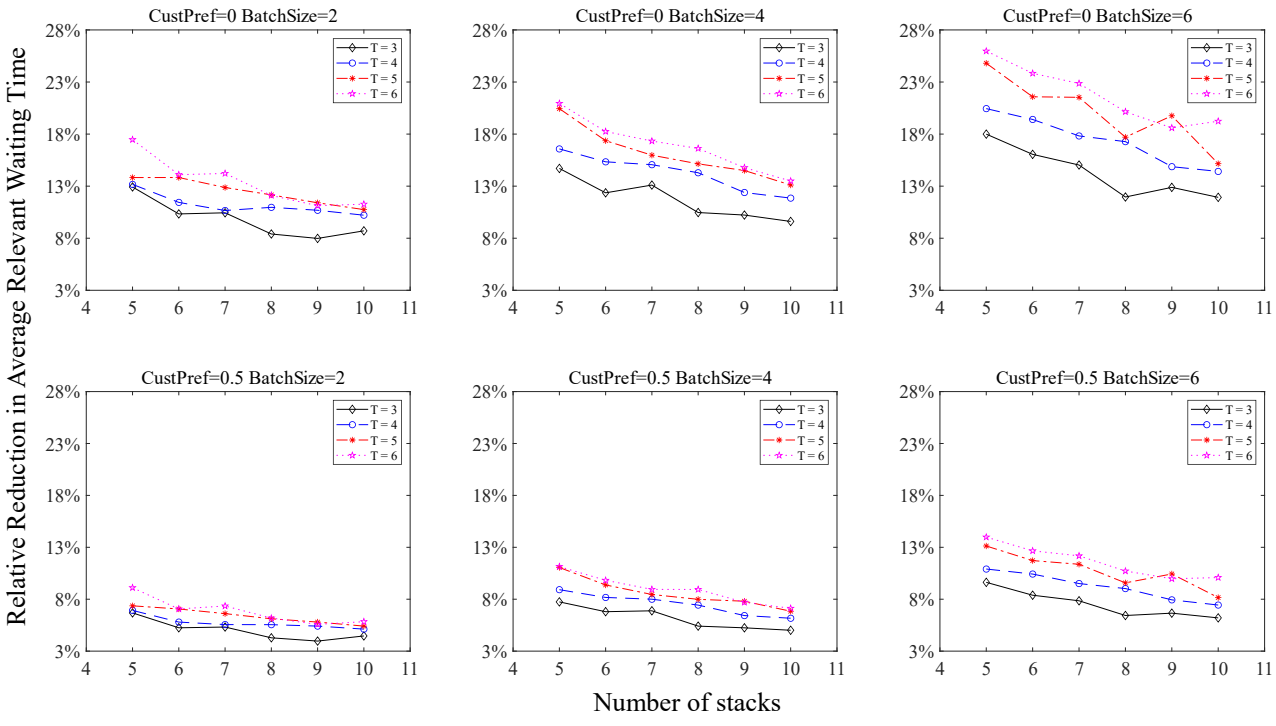


Fig. E.3 (a) Effect of the flexible service policy on average relevant waiting time for instances of 50% fill rate

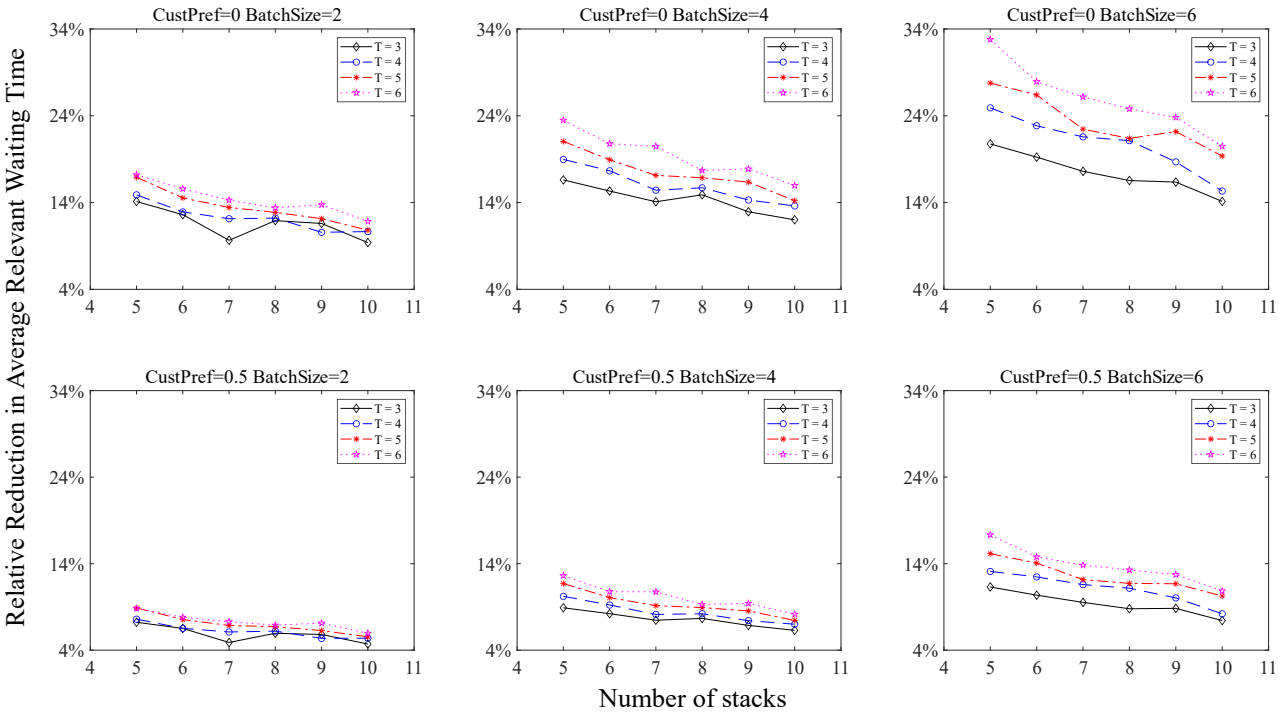


Fig. E.3 (b) Effect of the flexible service policy on average relevant waiting time for instances of 67% fill rate

E.4. Average truck turn time

Table E.1 Results of the average truck turn time for all instances under the flexible service policy

			Fill rate = 50%				*Fill rate* = 67%			
S	*B*	*P*	*T*=3	*T*=4	*T*=5	*T*=6	*T*=3	*T*=4	*T*=5	*T*=6
5	2	0	13.995	14.178	14.652	14.590	13.981	14.660	14.894	15.052
	2	0.5	21.709	21.915	22.396	22.393	21.731	22.467	22.764	22.975
	2	1	28.995	29.178	29.631	29.577	28.981	29.660	29.894	30.049
	4	0	17.676	17.402	18.330	18.276	17.370	18.551	18.984	18.983
	4	0.5	25.717	25.492	26.678	26.682	25.452	26.875	27.434	27.629
	4	1	32.676	32.402	33.315	33.276	32.370	33.525	33.956	33.967
	6	0	20.630	20.320	22.102	22.467	20.532	23.169	24.930	24.936
	6	0.5	29.109	28.927	31.329	31.828	29.179	32.577	34.865	35.674
	6	1	35.630	35.320	37.102	37.467	35.532	38.169	39.918	39.906
6	2	0	14.125	14.142	14.296	14.669	14.305	14.667	14.648	14.983
	2	0.5	21.804	21.859	22.055	22.454	22.038	22.437	22.459	22.851
	2	1	29.125	29.142	29.296	29.669	29.305	29.667	29.648	29.983
	4	0	17.944	17.594	18.031	18.821	17.892	18.495	18.653	19.230
	4	0.5	25.899	25.640	26.209	27.128	25.975	26.745	26.998	27.769
	4	1	32.944	32.594	33.031	33.821	32.892	33.495	33.653	34.221
	6	0	20.377	21.716	22.086	24.077	21.880	23.528	24.706	26.605
	6	0.5	28.733	30.411	30.961	33.468	30.576	32.725	34.531	36.959
	6	1	35.377	36.716	37.086	39.077	36.880	38.528	39.706	41.485
7	2	0	14.175	14.160	14.528	14.639	14.331	14.258	14.670	15.027
	2	0.5	21.852	21.829	22.277	22.295	22.003	21.976	22.455	22.861
	2	1	29.175	29.160	29.528	29.626	29.331	29.258	29.670	30.008
	4	0	17.613	17.892	18.410	18.411	18.174	17.818	18.652	19.758
	4	0.5	25.582	25.977	26.580	26.658	26.221	25.893	26.891	28.330
	4	1	32.613	32.892	33.410	33.408	33.174	32.818	33.637	34.751
	6	0	21.210	21.924	23.390	24.223	22.480	22.009	25.351	29.294
	6	0.5	29.559	30.513	32.495	33.544	31.099	30.922	34.755	39.858
	6	1	36.210	36.924	38.390	39.223	37.480	37.009	40.325	44.297
8	2	0	13.881	14.188	14.271	14.625	14.172	14.377	14.612	14.973
	2	0.5	21.514	21.891	21.999	22.247	21.889	22.115	22.374	22.779
	2	1	28.881	29.188	29.271	29.614	29.172	29.377	29.607	29.969
	4	0	17.154	17.964	17.976	18.730	17.706	18.142	18.487	19.407
	4	0.5	24.989	26.001	26.062	26.947	25.777	26.277	26.712	27.788
	4	1	32.154	32.964	32.976	33.730	32.706	33.142	33.478	34.399
	6	0	20.435	22.649	23.074	24.595	22.098	23.526	24.767	29.260
	6	0.5	28.524	31.295	31.759	33.710	30.625	32.603	33.968	39.559
	6	1	35.435	37.649	38.074	39.595	37.098	38.526	39.767	44.264
9	2	0	13.979	14.280	14.413	14.382	14.206	14.415	14.764	14.757
	2	0.5	21.610	21.981	22.111	21.918	21.917	22.125	22.521	22.542
	2	1	28.979	29.280	29.413	29.382	29.206	29.415	29.757	29.751
	4	0	17.411	17.937	18.368	18.120	17.885	18.235	19.012	18.968
	4	0.5	25.265	25.906	26.439	26.213	25.866	26.314	27.258	27.298
	4	1	32.411	32.937	33.368	33.120	32.885	33.235	34.006	33.959
	6	0	21.281	22.632	23.580	23.853	22.558	23.807	26.444	27.431
	6	0.5	29.524	31.075	32.547	32.706	31.083	32.673	36.034	37.316
	6	1	36.281	37.632	38.580	38.853	37.558	38.807	41.430	42.431
10	2	0	13.805	14.016	14.343	14.524	14.026	14.271	14.573	14.751
	2	0.5	21.436	21.686	22.020	22.074	21.692	21.963	22.287	22.502
	2	1	28.805	29.016	29.343	29.518	29.026	29.271	29.571	29.747
	4	0	17.383	17.450	18.151	18.725	17.516	17.854	18.340	19.175

4	0.5	25.208	25.376	26.165	26.799	25.442	25.880	26.421	27.433
4	1	32.383	32.450	33.151	33.720	32.516	32.854	33.338	34.173
6	0	20.750	21.174	23.791	25.256	21.192	22.956	25.013	27.868
6	0.5	28.878	29.498	32.363	34.376	29.493	31.476	34.084	37.408
6	1	35.750	36.174	38.791	40.256	36.192	37.956	40.005	42.872

Appendix F. Results of three sets of customer preference scenarios

Table F.1 Comparison between three sets of customer preference scenarios for small batches and 50% fill rate

T	*S*	*C*	'100%' preference scenario				'50%' preference scenario				Heterogeneous preference scenario			
			Solved	CPU(s)	Rel	AveWait	Solved	CPU(s)	Rel	AveWait	Solved	CPU(s)	Rel	AveWait
3	5	8	√	0.04	1.267	2.992	√	0.03	1.478	3.207	√	0.02	1.475	3.205
	6	9	√	0.03	1.433	3.126	√	0.03	1.581	3.303	√	0.03	1.593	3.306
	7	11	√	0.03	2.433	3.170	√	0.03	2.654	3.356	√	0.04	2.638	3.348
	8	12	√	0.03	2.033	2.883	√	0.03	2.169	3.014	√	0.04	2.158	3.017
	9	14	√	0.05	2.767	2.976	√	0.04	2.885	3.106	√	0.05	2.885	3.111
	10	15	√	0.06	3.000	2.800	√	0.06	3.093	2.930	√	0.07	3.088	2.939
4	5	10	√	0.04	2.633	3.180	√	0.03	2.855	3.411	√	0.04	2.862	3.418
	6	12	√	0.05	3.267	3.144	√	0.05	3.466	3.351	√	0.06	3.475	3.355
	7	14	√	0.07	3.733	3.148	√	0.05	3.941	3.324	√	0.08	3.944	3.340
	8	16	√	0.15	4.300	3.192	√	0.20	4.559	3.390	√	0.31	4.533	3.387
	9	18	√	0.37	5.333	3.274	√	0.34	5.523	3.474	√	0.76	5.531	3.477
	10	20	√	0.27	5.767	3.010	√	0.80	6.015	3.181	√	2.65	6.003	3.187
5	5	13	√	0.08	4.500	3.621	√	0.19	4.884	3.900	√	0.32	4.856	3.884
	6	15	√	0.20	5.200	3.307	√	2.98	5.544	3.553	√	34.94	5.552	3.565
	7	18	√	0.28	6.200	3.519	√	0.77	6.573	3.777	√	2.29	6.562	3.783
	8	20	√	1.43	7.200	3.267	√	8.16	7.516	3.482	√	34.67	7.501	3.484
	9	23	√	*5.00*	*8.414*	*3.388*	29/30	68.25	8.696	3.606	284/300	151.57	8.700	3.615
	10	25	√	*7.33*	*8.931*	*3.321*	29/30	49.37	9.238	3.517	285/30**0**	158.48	9.235	3.529
6	5	15	√	1.25	6.400	3.560	√	5.06	6.999	3.886	√	15.69	7.008	3.897
	6	18	√	0.99	7.233	3.670	√	6.18	7.728	3.959	299/30**0**	65.82	7.727	3.963
	7	21	27/30	*44.46*	*8.455*	*3.506*	22/30	160.87	8.923	3.787	208/300	*260.84*	*8.843*	*3.795*
	8	24	25/30	*23.52*	*9.333*	*3.500*	22/30	*69.554*	*9.758*	*3.725*	208/300	184.58	9.751	3.734
	9	27	24/30	*48.71*	*10.389*	*3.235*	18/30	104.47	10.538	3.419	173/300	194.41	10.530	3.428
	10	30	23/30	*38.75*	*11.333*	*3.404*	18/30	108.52	11.572	3.580	-	-	-	-