European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing, Transportation and Logistics

Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem

Bernard G. Zweers[a,b,*], Sandjai Bhulai[b,a], Rob D. van der Mei[a,b]

[a] Centrum Wiskunde & Informatica, Stochastics, Science Park 123, 1098XG Amsterdam, the Netherlands
[b] Vrije Universiteit Amsterdam, Department of Mathematics, De Boelelaan 1105, 1081HV Amsterdam, the Netherlands

ARTICLE INFO

Article history:
Received 24 July 2019
Accepted 28 November 2019
Available online xxx

Keywords:
Logistics
Stochastic Container Relocation Problem
Container pre-marshalling
Branch-and-bound
Local search

ABSTRACT

In container terminals, containers are often moved to other stacks in order to access containers that need to leave the terminal earlier. We propose a new optimization model in which the containers can be moved in two different phases: a pre-processing and a relocation phase. To solve this problem, we develop an optimal branch-and-bound algorithm. Furthermore, we develop a local search heuristic because the problem is NP-hard. Besides that, we give a rule-based method to estimate the number of relocation moves in a bay. The local search heuristic produces solutions that are close to the optimal solution. Finally, for instances in which the benefits of moving containers in the two different phases are in balance, the solution of the heuristic yields significant improvement compared to the existing methods in which containers are only moved in one of the two phases.

1. Introduction

Shipping containers are the main way of transporting goods around the world. About 75% of the volume of the total trade is carried by sea and about half of this trade is shipped in containers (Lee & Song, 2017). A key part of this global supply chain are container terminals, at which containers are transshipped between different modes of transportation. As a result, for each part of the transportation, the most efficient mode of transportation can be used. Deep-sea vessels are used to ship thousands of containers from one continent to another and they only visit a few deep-sea ports. At these deep-sea ports, the containers are transshipped to barges or trains for the inland transportation. The barges or trains bring from tens to a few hundred containers to an inland container terminal. Afterward, a truck is used to ship the container from the inland container terminal to its final destination.

Although the terminals facilitate the shipment of containers on different modes, the handling of a container at a terminal imposes extra costs. To make things worse, containers are most of the times moved multiple times inside the terminal. After a container arrives at the terminal, it is temporarily stored in a container yard because the different modes of transportation are often not synchronized. In order to save space, the containers are stacked on top of each other, but the handling equipment of a terminal can only access the top container. Ideally, each container that needs to leave the terminal is located at the top of its stack. However, when a container arrives at the container terminal it is often not known when exactly it needs to leave the terminal. Consequently, it frequently occurs that when a container needs to leave the terminal, other containers are stacked on top of that target container. These blocking containers will then need to be relocated to other stacks. These moves are called *relocation moves* and since they impose extra costs and delays, they need to be prevented as much as possible.

At an inland terminal, all outbound import containers need to be delivered to their final destinations. This last part of the transportation is almost always done with a truck and in order to serve as many trucks as possible on a day, a terminal is interested in reducing the turnaround time of a truck. One way to reduce the truck's turnaround time is to perform fewer relocation moves because the relocation moves are performed when a truck is waiting at the terminal. At an inland terminal, the crane to handle containers is more often idle than at a deep sea terminal because fewer containers are transshipped via an inland terminal. If the crane could do some pre-processing when it is idle to reduce the number of relocation moves, then the turnaround times of trucks could be reduced. Consequently, fewer trucks might be needed to transport all containers. An inland terminal in the port of Amsterdam is facing the problem which pre-processing move to perform in order to reduce the number of relocation moves. To answer that question, we present in this paper a new optimization problem, which we

* Corresponding author at: Centrum Wiskunde & Informatica, Stochastics, Science Park 123, 1098XG Amsterdam, the Netherlands.
E-mail address: b.g.zweers@cwi.nl (B.G. Zweers).

https://doi.org/10.1016/j.ejor.2019.11.067
0377-2217/

call the *Stochastic Container Relocation Problem with Pre-Processing* (SCRPPP).

The contribution of this work is fourfold. First, we introduce a new optimization problem faced by a real inland container terminal. Second, we develop a heuristic to solve this problem and third, we present an optimal algorithm for this problem. Finally, a method to estimate the number of relocation moves for a given bay is presented.

The SCRPPP is closely related to other operational problems in container terminals. Hence, we start by describing the relevant literature for these problems in Section 2. Afterward, we will give a detailed description of the SCRPPP in Section 3. In Section 4, we prove that the SCRPPP is NP-hard and derive bounds for the optimal solution and the maximum number of containers in a bay. In order to solve the SCRPPP, we present in Section 5 both a local search heuristic and an optimal algorithm. The optimal and heuristic solutions are compared with each other in Section 6. Finally, in Section 7, we draw conclusions and present directions for further research.

2. Literature review

At a container terminal, many different types of optimization problems are faced. For a general overview of these problems we refer to (Stahlbock & Voß, 2008) and (Steenken, Voß, & Stahlbock, 2004). The focus of this paper is one specific type of problem, namely stacking policies for containers. These stacking policies can be applied to a yard with only inbound or outbound containers or a combination of the two. For overview articles of the different types of stacking problems we refer to Carlo, Vis, and Roodbergen (2014), Caserta, Schwarze, and Voß (2011a), and Lehnfeld and Knust (2014). In this work, we will only focus on the stacking of outbound containers. There are two main problems concerning the stacking of outbound containers: (i) the Container Relocation Problem (CRP), sometimes also called blocks relocation problem, and (ii) the Container Pre-Marshalling Problem (CPMP). To be more precise, we will study a generalization of the CRP, which is called the Stochastic Container Relocation Problem (SCRP). Since our problem can be seen as a combination of the SCRP and the CPMP, we will discuss the relevant literature for the two problems below.

In the CRP, each container has a unique label that indicates the order in which the containers are picked up by a truck. Containers that are on top of a container that is retrieved need to be relocated to other stacks. The objective of the CRP is to use as few of these relocation moves as possible. The CRP was introduced by Kim and Hong (2006) and is proven to be NP-hard (Caserta, Schwarze, & Voß, 2012). As a result of this hardness, the literature concerning the container relocation problem can be divided into two different categories. The first branch of the literature deals with finding optimal solutions for the CRP and the second stream of research focuses on heuristics for the CRP. One way to solve the CRP to optimality is to use integer programming. Caserta et al. (2012) were the first to introduce such a formulation. However, that formulation needed hours to solve instances of twenty to thirty possible locations of containers. In Zehender, Caserta, Feillet, Schwarze, and Voß (2015), the formulation of Caserta et al. (2012) is improved, but it is still not able to solve realistic instances.

Heuristics have been proposed for the CRP to solve larger instances in reasonable time. In Caserta, Voß, and Sniedovich (2011b), a heuristic for the CRP that is based on dynamic programming is presented. As the number of states that are examined is restricted this heuristic runs fast, but a heuristic that gives better solutions is a rule-based heuristic described in Caserta et al. (2012). Another heuristic in which a restricted number of states is considered is presented in Wu and Ting (2012). In this article, the beam search procedure is applied. Also, a few meta-heuristics have been developed for the CRP. In Jovanovic, Tuba, and Voß (2019b) the CRP is solved using ant colony optimization. A genetic algorithm is used in Hussein and Petering (2012) to solve a CRP variant in which the energy consumption is minimized. The energy consumption depends on the weight of a container, so heavy containers should not move too far. They use the genetic algorithm to find the best parameter settings of a constructive heuristic.

In Ji, Guo, Zhu, and Yang (2015), a variant of the CRP is introduced that also incorporates loading plans of ships. In this problem, a stowage plan of a ship is given and the loading sequence of the containers has to be decided such that the number of relocations in the container yard is minimized. Ji et al. (2015) use a genetic algorithm to find good solutions for this problem. In Jovanovic, Tanaka, Nishi, and Voß (2019a), a GRASP heuristic is proposed for the same problem and this heuristic gives significantly better solutions than the method of Ji et al. (2015).

The SCRP, which is a generalization of the CRP was introduced by Zhao and Goodchild (2010). In the SCRP, the exact order in which containers are retrieved is not known anymore. It is only given in which time interval a container is retrieved, but multiple containers could have the same interval. The retrieval order of the containers in the same time interval is a uniform random permutation. Using a simple heuristic, Zhao and Goodchild (2010) conclude that the value of information is important for this problem. In Ku and Arthanhari (2016), a more advanced heuristic called the Expected Reshuffling Index is introduced. The idea behind this heuristic is to calculate a score for every stack based on the expected number of reshuffles needed for that stack. A container is relocated to the stack with the smallest number of reshuffles. In Galle, Manshadi, Borjian Boroujeni, Barnhart, and Jaillet (2018), another heuristic is introduced, which is called the Expected Min-max (EM) heuristic. In the EM heuristic, containers are relocated to stacks in which they are relocated as late as possible. Besides the EM heuristic, in Galle et al. (2018) an optimal formulation for the SCRP using decision trees is presented. For small instances, the optimal solution can be computed which is close to the solution of the EM heuristic.

The CPMP was introduced by Lee and Hsu (2007) and its goal is to reshuffle the containers before any container is retrieved, in such a way that all containers can be retrieved without any further relocations. The objective is to use as few moves as possible to obtain such a layout. In Lee and Hsu (2007), the CPMP is modeled as a multi-commodity flow network, which can be solved to optimality using an integer program. Nevertheless, they are only able to solve small instances to optimality. In Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2012), an A^* algorithm is proposed to solve the CPMP to optimality, but this method can also only solve relatively small instances. The method of Expósito-Izquierdo et al. (2012) is improved by Tierney, Pacino, and Voß (2017) using the lower bounds for the CPMP introduced by Bortfeld and Forster (2012). In Parreño-Torres, Alvarez-Valdes, and Ruiz (2019), eight different integer linear formulations are presented, which give the best current mathematical models for the CPMP. Nevertheless, the branch-and-bound algorithm of Tanaka and Tierney (2018) is the current state-of-the-art algorithm to solve the CPMP to optimality. However, for larger instances, the running time is still often more than an hour.

The CPMP has never been proven to be NP-hard, but since none of the optimal algorithms produces a fast solution also heuristics have been developed. One of the heuristics that is proposed is the Lowest Priority First Heuristic (LPFH) (Expósito-Izquierdo et al., 2012). The idea behind the LPFH is to place the container with the largest time frame, i.e., the lowest priority, first in a rightly ordered stack. Afterward, the container with the second lowest priority is placed in a well-ordered stack, and so on. In Jovanovic, Tuba, and

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

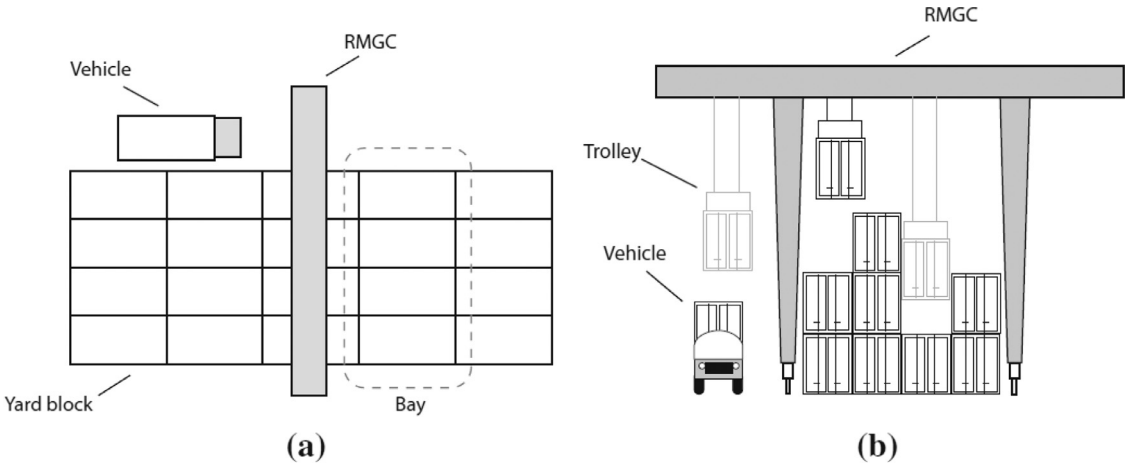


Fig. 1. The layout of an RMGC and a container yard (Tierney et al., 2017).

Voß (2017), an improved version of the LPFH is given. In Bortfeld and Forster (2012), a tree search procedure is introduced to solve the CPMP.

To the best of our knowledge, there is only limited work dealing with uncertainty in the CPMP, namely Rendl and Prandtstetter (2013) and Tierney and Voß (2016). These works study the Robust Container Pre-Marshalling Problem in which for each container an interval in which the containers could be retrieved is given. The goal is to place containers such that the latest possible departure time of a container is always earlier than the earliest possible departure time of a container underneath it. The difference between the intervals in this problem and in the SCRPPP is that here the interval width is container-dependent. In Rendl and Prandtstetter (2013), this problem is solved using constraint programming and in Tierney and Voß (2016) an IDA* heuristic is introduced that outperforms the earlier work of Rendl and Prandtstetter (2013) both in computation time and solution quality.

3. Problem description

The SCRPPP is inspired by a problem faced by an inland terminal in the port of Amsterdam. At the end of the day, it is known which containers will leave the terminal the next day. The containers that are going to their final destination will be picked up by a truck. For each container, it is known in which time period a truck will arrive to pick it up. However, due to external factors this time period is not very precise and multiple trucks may arrive in the same time interval. As a consequence, multiple containers are picked up in the same time interval and we do not know the arrival order of the trucks in that period. Furthermore, at the end of the day, it is usually also less busy at the terminal than during the course of the day. As a result, the crane driver is sometimes idle and could do already some moves that reduce the number of moves of the next day. However, he or she might not always have time to do all pre-marshalling moves from the CPMP. On top of that, it might not be motivating to move many containers in order to prevent only one relocation move.

In order to model the situation sketched above, in the SCRPPP containers can be moved in two different phases: the *pre-processing* phase and the *relocation* phase. The pre-processing moves can only be executed *before* any of the containers is retrieved, whereas the relocation phase starts the moment the first container is retrieved. The relocation moves are equivalent to the moves executed in the SCRP. Although the pre-processing moves might look similar to the pre-marshalling moves in the CPMP, there is one major difference: the containers might still need to be relocated after we finish the pre-processing phase. This novel degree of freedom could be used to reduce the total number of moves, but it also increases the problem's complexity. Since one has to decide when to stop with the pre-processing phase.

To describe the SCRPPP in more detail, we will first describe the operational process at a container terminal. In container terminals, containers are stored in a rectangular yard, see Fig. 1(a). Most often, the equipment to handle the containers in a yard is a Rail Mounted Gantry Crane (RMGC). In Fig. 1, an illustration of a container yard and an RMGC is given. As shown in Fig. 1(a), the RMGC is positioned above a single row of containers which is called a *bay*. Using the trolley that is attached to the crane, see Fig. 1(b), all stacks of containers in that bay can be accessed. Obviously, only the top container of each stack can be retrieved by the trolley. If the trolley has retrieved a container it can either move to another stack to place the container there or it can move to the end of the RMGC to place the container on a truck. The yard in Fig. 1 corresponds to a yard that is positioned parallel to the quay. In some terminals, the lay-out of the yard is slightly different and the yard is positioned perpendicular to the quay. In the latter type of terminals, a vehicle cannot be positioned next to every bay as in Fig. 1(b), but it can only receive at the first and/or last bay of the yard. That means that if a containers is retrieved, the entire crane needs to move to the end of yard. In the following explanation, we assume for simplicity a yard that is positioned parallel to the quay but our methods also apply to a yard that is perpendicular to the quay.

In case a container needs to be relocated to another bay, the entire RMGC needs to move. Compared with moving only the trolley this is time-consuming. On top of that, some terminals forbid for safety reasons a crane to move when it is carrying a container (Lee & Hsu, 2007). Hence, in the SCRPPP it is assumed that no container is relocated to a different bay. Consequently, the SCRPPP is a two-dimensional problem, similar to Fig. 1(b). The height of the RMGC imposes a maximum height to a stack of containers because the trolley needs to be able to move with a container from one side of the bay to the other. For instance, in Fig. 1(b) the maximum height of a stack is three containers. On top of that, the width of RMGC also imposes a maximum on the number of stacks in a yard. Hence, the total number of containers that can be placed in a bay is limited. We refer to the available positions of containers as *slots*. The bay given in Fig. 1(b) contains twelve slots because the maximum number of stacks is four that all have a maximum height of three.

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

In order to have a good balance between the current practice at container terminals and computational tractability five assumptions are made for the SCRPPP:

1. All containers in a bay will leave the bay before any new containers arrive.
2. A container may only be moved in the relocation phase if it is blocking the container that needs to be retrieved.
3. For each container, the time interval in which it is picked up is known, but the retrieval order inside an interval is a random uniform permutation.
4. The cost of moving a container from one stack to the other does not depend on the stack to which a container is moved.
5. It is physically possible to stack every container on every other container.

Assumption 1 is based on the fact that many terminals have designated areas for inbound and outbound containers. Moreover, when containers are unloaded from a ship, they are mostly all placed in empty bays. Therefore, a bay with outbound containers will have no new containers arriving before it is empty. The problem that arises when this assumption is not made is called the *dynamic container relocation problem* (Akyüz & Lee, 2014).

To the best of our knowledge, Assumption 2 is always imposed on the SCRP. Besides the fact that this assumption significantly decreases the problem's complexity, it also represents the current practice at container terminals (Caserta et al., 2012). In the deterministic CRP, Assumption 2 is sometimes not made and the resulting problem is called the unrestricted CRP. We refer to Tanaka and Mizuno (2018) for a recent article about the unrestricted CRP.

Assumption 3 is inspired by the situation in which a terminal has a truck appointment system (Ku & Arthanhari, 2016). In such a system, trucks make an appointment to pick up containers in a certain time interval or also called time frame. The terminal then knows which containers will be picked up in which interval, but it does not know which one of the trucks in one interval will be the first to arrive and pick up a container. The moment at which the retrieval order of an interval is revealed gives two different models. In the first model, the order of the other containers in an interval remains unknown if the first container of an interval is revealed. This model is called the *online model* (see, e.g., Ku & Arthanhari, 2016 and Zhao & Goodchild, 2010). Whereas, in the *batch model*, at the beginning of each interval the exact order of all containers in that single interval is revealed. The batch model is more suitable for larger terminals in which the waiting time of a truck is often larger than the time length of the interval (Galle et al., 2018). As our problem setting is more applicable to less busy ports, we choose to use the online model.

A consequence of Assumptions 2 and 3 is that it is easy to check whether a container will be moved in the relocation phase. A container will only be moved in the relocation phase if a container that is positioned below it is retrieved earlier. If all time frames of containers below a container are larger than its own time frame, it will not need to be relocated. Otherwise, the expected number of relocation moves of a container is always larger than 0. We say that a container for which no relocation moves are needed is *well-placed* or *correctly placed*.

Another consequence of Assumption 3 is that the number of relocation moves that need to be executed to retrieve all containers is stochastic. The variability of the number of relocation moves is more than one might expect at first sight. For example, consider a stack that consists out of two containers from the same time interval. If the top container is the first to be retrieved and the bottom container the last, no relocation moves are needed. Whereas if the bottom container is the first to be retrieved, one relocation move is needed for the top container. Hence, with probability $\frac{1}{2}$ no relocation moves are needed and with probability $\frac{1}{2}$ one relocation move is needed for the containers in that interval. On top of that, note that the fact that the retrieval order of containers in one specific interval is stochastic does not only influence the expected number of relocation moves needed to retrieve the containers from that interval. The relocation moves that are performed in one time interval influence the layout of the bay after that interval has ended. Because of Assumption 2, it is only allowed to move containers that are on top of the target container. Therefore, the retrieval order of the containers inside an interval has a big influence on the order in which relocation moves are performed. Consequently, it might be that each retrieval order of an interval results in a different layout after the interval is finished. As a result, the retrieval order of the first interval might influence the number of relocation moves needed for the last interval and thus, the total number of relocation moves can vary across a wide range of values.

Assumption 4 is based on the fact that the time needed to pick up and release a container with a trolley is considerably larger than the time needed to move the trolley. As a result, the stack to which a container is relocated does not have a big influence on the total relocation time of a container. There are a few articles in the literature in which Assumption 4 is relaxed and in which the objective is to minimize a weighted average of the number of relocation moves and the total working time of the crane (see, e.g., Lin, Lee, & Lee, 2015 and da Silva Firmino, de Abreu Silva, & Times, 2019). In a recent study, Voß and Schwarze (2019) show that if the crane's working time is the objective that is minimized, then also the number of relocation moves is close to the minimum number of relocation moves. A consequence of Assumption 4 is that the order of the stacks is not relevant and any permutation of the stacks can be treated as if it were the same instance.

Another consequence of Assumption 4 is that only the number of moves is relevant in a solution. In the SCRPPP there are two different types of moves: moves in the pre-processing phase and moves in the relocation phase. We are interested in minimizing a weighted average of these two types of moves. The moves in the pre-processing phase are executed when the crane would otherwise be idle, whereas during the relocation moves a truck is waiting. Therefore, the weight assigned to pre-processing moves is lower than the weight for relocation moves. We normalize the weight for relocation moves to 1 and set the weight for pre-processing moves to $0 < \alpha < 1$. The value of α can be seen as an indication of how inclined someone is to perform pre-processing moves. If, for instance, α is set to $\frac{1}{4}$, then one is willing to perform at most four pre-processing moves in order to prevent one relocation move.

Since containers have different sizes, Assumption 5 is not evident for general containers because a container of forty feet cannot be stacked on top of a single container of twenty feet. However, Assumption 5 is an assumption that is not restricting from a practical perspective because most terminals locate containers with different sizes in different bays. Moreover, in case there would be certain containers on which we cannot stack a container, the SCRPPP is easier: when we need to determine to which stack a container is moved, we could simply ignore the stacks in which we cannot stack the container.

After the assumptions for the SCRPPP, we will now give the formal objective of the SCRPPP. Let us call the initial bay before the pre-processing phase B. We can denote the movement of the top container from stack o to stack d as the pair (o, d). If p is the number of pre-processing moves executed, the set of all pre-processing moves can be denoted by: $P = \{(o_1, d_1), (o_2, d_2), \ldots, (o_p, d_p)\}$. The bay that is obtained by the pre-processing moves P will be called $B(P)$. Assume that there are n different retrieval orders of the containers and let σ_i be one of those orders. If in a bay B the relocation moves are performed according to a certain policy π, then the number of relocation moves for the retrieval order σ_i is given

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

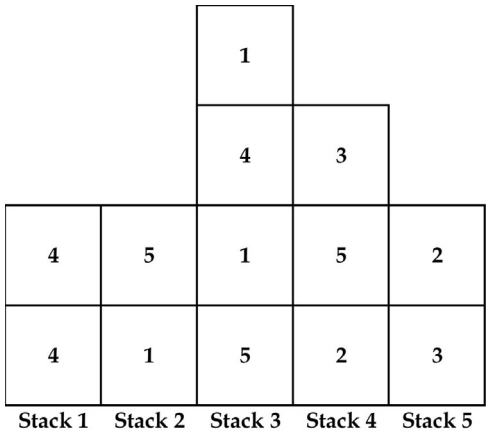


Fig. 2. Example of an instance for the SCRPPP with 5 stacks that have a maximum height of 4.

by $R_\pi(B, \sigma_i)$. A policy π can be any rule that decides, given the complete bay and a target container to which container a stack needs to be relocated. The goal of the SCRPPP is to find the pre-processing moves P and the policy π such that the weighted sum of the number of pre-processing moves and the expected number of relocation moves is minimized. In other words, the goal is to find the following minimum:

$$\min_{P,\pi} \alpha|P| + \frac{1}{n}\sum_{i=1}^{n} R_\pi(B(P), \sigma_i). \tag{1}$$

The part $\alpha|P|$ in Eq. (1) is equivalent to the costs for the pre-processing moves. Since $R_\pi(B, \sigma_i)$ is the number of relocation moves for one specific retrieval order σ_i, the expected number of relocation moves is calculated via $\frac{1}{n}\sum_{i=1}^{n} R_\pi(B(P), \sigma_i)$. At first sight one might think that it is beneficial to perform that many pre-processing moves such that no relocation moves are needed because pre-processing moves have a lower weight than relocation moves. However, relocation moves have two advantages compared to pre-processing moves. The first advantage is that during the pre-processing phase all original containers are still in the bay, while when a relocation move is executed some containers have already left the bay. The second advantage of a relocation move is that when the container needs to be relocated at least some part of the retrieval order inside an interval is known. On the contrary, when a pre-processing move is done, no information about the retrieval order inside an interval is known.

If α is close to 1, the two benefits of the relocation moves outperform the advantage of the pre-processing moves which have a slightly smaller weight. Hence, no pre-processing moves will be performed and the problem is similar to the SCRP. On the other hand, if α is close to 0 as many pre-processing moves as needed will be performed in order to prevent any relocation move and the problem is equivalent to the CPMP. However, for more moderate values of α it is hard to find the right balance between pre-processing moves and relocation moves.

Running example

Throughout this paper, an example will be used to illustrate the SCRPPP and its solution methods. This instance for the SCRPPP is given in Fig. 2 and the viewpoint in this figure is the same as in Fig. 1(b). The bay in Fig. 1 consists out of five stacks that all have a maximum height of four containers. The numbers inside each container represent the interval in which the container is retrieved. We will number the stacks from left to right, so the leftmost stack is stack number 1 and the rightmost stack is stack number 5. Assume for the moment that no pre-processing moves are performed.

Table 1
Notation for the SCRPPP.

B	Specific layout of a bay
$\mathcal{S}$	Set of all stacks in a bay
$B(s_1, s_2)$	Layout of bay B after the top container of stack s_1 has moved to stack s_2
Z	Largest time frame in a bay
H	Maximum height of a stack
$\mathcal{C}$	Set of containers in a bay
C	Number of containers in a bay
$n(s)$	Number of containers in stack s
$l(s)$	Smallest time frame of stack s
$h(s)$	Largest time frame of stack s
$s(c)$	Stack of container c
$u(c)$	Smallest time frame of containers underneath c
$t(c)$	Time frame of container c
$q(c)$	Category of container c
$p(c)$	Position of a container c in stack $s(c)$ 1 is the lowest and H the highest position
UB	Upper bound for the optimal solution
LB	Lower bound for the optimal solution
$f(B)$	Estimated of expected number of relocation moves in bay B

Then one of the three containers with time frame 1 is the first container to be retrieved. If the top container of the middle stack is the first container to be retrieved, no relocation moves are needed because no other containers are on top of it. However, the bottom container of the second stack is blocked by the container with time frame 5. So if this container is the first to be retrieved, the container with time interval 5 has to be relocated to another stack. If one would use the optimal policy for the relocation moves, the expected number of relocation moves for the bay in Fig. 2 is $6\frac{1}{3}$.

Let us now consider a situation in which we perform four pre-processing moves to the bay in Fig. 2, namely $P = \{(4,1),(4,5),(4,1),(2,1)\}$. In Fig. 3, the bays after all these four pre-processing moves are shown. The idea of the first three pre-processing moves is to empty the fourth stack such that in the fourth pre-processing move the container with time frame 5 from the second stack can be placed in the fourth stack. Note that this container with time interval 5 was not placed correctly in stack 2, but that it is well-placed in the fourth stack. If we apply the optimal policy for the relocation moves to the bay in Fig. 3(d), the expected number of relocation moves is 3. Hence, the objective function for the bay in Fig. 2 to which we apply pre-processing moves $P = \{(4,1),(4,5),(4,1),(2,1)\}$ and the optimal relocation policy is $4\alpha + 3$. As the objective function when no relocation moves were performed equals $6\frac{1}{3}$, the moves P are beneficial as long as $\alpha \leq \frac{3\frac{1}{3}}{4} = \frac{5}{6}$.

4. Bounds and complexity

In this section, we present bounds regarding the maximum number of containers in a bay and the optimal solution. However, first of all, we will show in Section 4.1 that the SCRPPP is NP-hard. In Section 4.2, we will give a bound on the maximum number of containers that can be in a bay for a feasible solution of the SCRPPP. Moreover, it is shown when it is possible to move a container in the correct position in the pre-processing phase. In Section 4.3, an upper bound for the optimal solution of the SCRPPP is given and a lower bound for the optimal solution is derived in Section 4.4. Throughout this paper, we will need some notation which is introduced in Table 1.

4.1. Complexity

In Caserta et al. (2012), the CRP is proven to be NP-hard. In this proof, the decision problem of *Mutual Exclusion Scheduling* (MES) is reduced to an instance of the decision version of the CRP. Solving

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

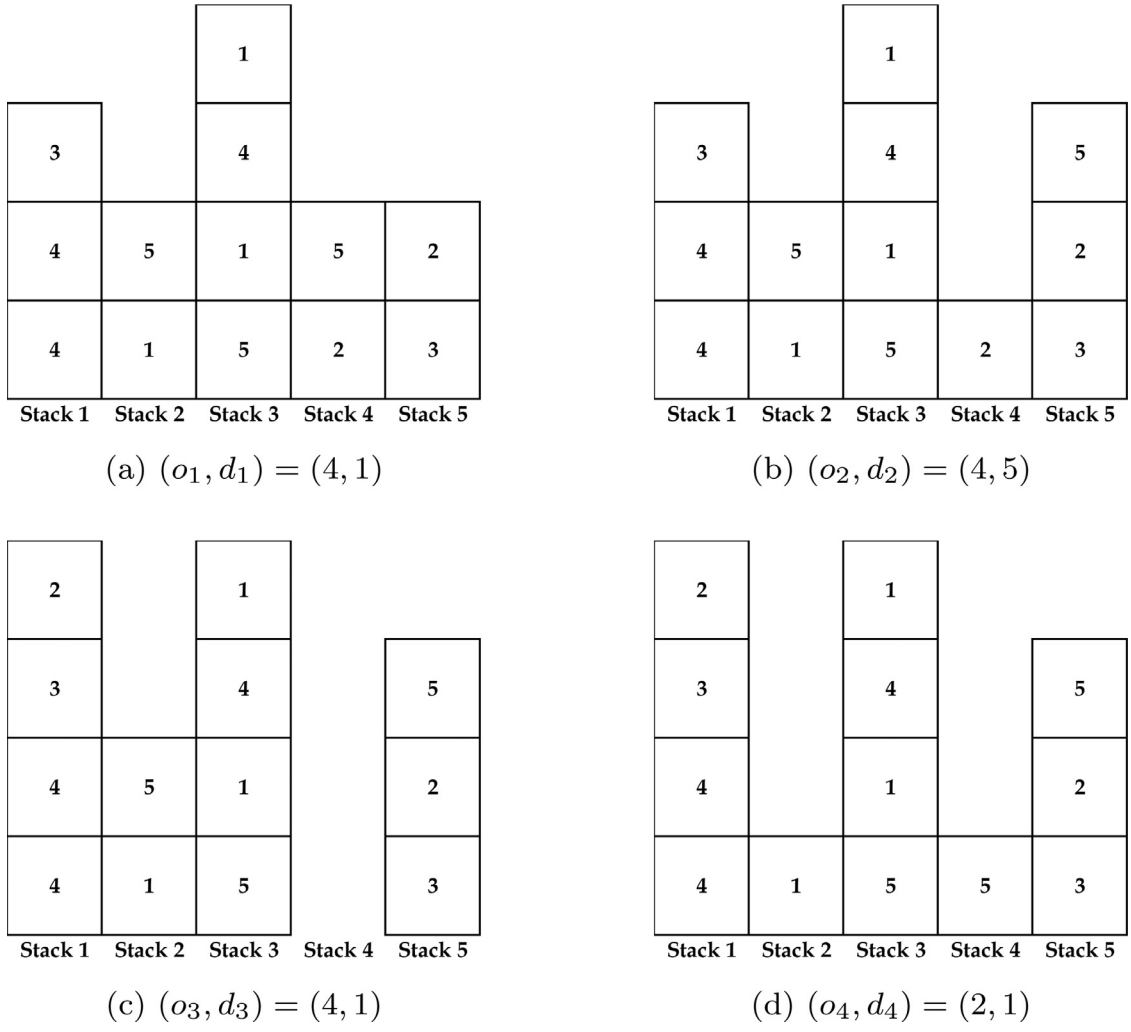


Fig. 3. Four pre-processing moves for the bay of Fig. 2.

that instance is shown to be equivalent to a yes-instance of the MES by Caserta et al. (2012). That specific instance is also an instance for the SCRPPP and it is trivial to show that it can be solved in n pre-processing moves and no relocation moves if and only if it corresponds to a yes-instance of the MES. Hence, deciding if the objective function of the SCRPPP equals αn is equivalent to finding a yes-instances of the MES. Therefore, the SCRPPP is also NP-hard.

4.2. Feasibility

If all stacks in a bay contain the maximum number of containers, it is likely that the SCRPPP does not have a feasible solution. Since no pre-processing moves are possible and if relocation moves are needed to retrieve the first container, then there are no empty slots for the containers that need to be relocated. In Lemma 1, we give a bound on the maximum number of containers that can be in a bay in order for the SCRPPP to have a feasible solution.

Lemma 1. *If the number of containers C is bounded by $C \leq SH - (H-1)$, then there exists a feasible solution for the SCRPPP.*

The proof of Lemma 1 follows directly from the condition in Caserta et al. (2012) that the number of containers in a bay does not exceed $SH - (H-1)$ is sufficient for feasibility of the CRP. The stochasticity of the SCRP does not influence this bound and since not performing any pre-processing moves is feasible for the SCRPPP, this bound also applies for the SCRPPP.

The bound given in Lemma 1 only considers the moves executed in the relocation phase. However, if there are $SH - (H-1)$ containers in a bay, it might not be possible to move a container in the pre-processing phase such that it becomes well-placed. As an example, consider a bay in which all stacks but one have reached their maximum height and the other stack contains only a single container. In this bay, exactly $SH - (H-1)$ containers are situated. If one of the completely filled stacks has a container with the largest time frame Z as its top container, an empty stack is needed before this container can be well-placed. The reason for that is that this container with time frame Z can only be well-placed at the bottom of a stack. Since there are only $H-1$ free slots in the bay, it is impossible to get an empty stack and correctly place the container with time frame Z. However, if we have H available slots in a bay, it is possible to place any container to the bottom of a stack as is shown in Lemma 2.

Lemma 2. *If there are H empty slots in a bay B with $S > 2$ stacks, it is possible to move every container to the bottom of each stack in the pre-processing phase.*

This lemma holds because if there are H free slots, it is always possible to arrange the containers in such a way in the pre-processing phase that one stack is entirely empty. Moreover, if there are more than two stacks, we can make sure that container c is the top container of a stack. Since we have shown above that it might not be possible to move a container to the bottom of a stack

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

if there are $H-1$ free slots in a bay, we know that the bound given in Lemma 2 has to be tight. A consequence of Lemma 2 is that if the number of containers in a bay with more than two stacks is limited by $(S-1)H$, then each container can become well-placed.

Corollary 1. *If a bay B with $S>2$ stacks has at most $C \leq (S-1)H$ containers, every container could be moved in the pre-processing phase such that it becomes well-placed.*

Proof. By Lemma 2 we know that there is always an option to place a container at the bottom of a stack if there are H free slots in a bay with more than two stacks. Since a container at the bottom of a stack is always well-placed, we know that each container can become well-placed if the number of free slots is at least H. If the number of containers in a bay is bounded from above by $(S-1)H$, the number of free slots should be at least H. □

In Lemma 2, we have shown that there always exists a feasible solution to move a container from one stack s to the bottom of another stack s' if there are H free slots in the bay. However, under that condition, one might need to move containers from a third stack s''. In the heuristic we give in Section 5.1, we would like to move a container in the pre-processing phase from a stack s to a correct position in stack s' without moving containers from other stacks. In Lemma 3 and Corollary 2, we prove that that is possible if there are $2(H-1)$ free slots.

Lemma 3. *If a bay B has $2(H-1)$ empty slots, each container in stack $s \in \mathcal{S}$ can be moved in the pre-processing phase to the bottom of stack $s' \in \mathcal{S}$ without moving containers from the stacks $\mathcal{S} \setminus (\{s\} \cup \{s'\})$.*

Proof. Let container c be the container that is moved in the pre-processing phase such that it is placed at the bottom of stack s'. As a result of Assumption 4, we can reshuffle the stacks in any order without adjusting the problem. Thus, if container c is already placed at the bottom of a stack, then we can shuffle the stacks such that container c is placed at the bottom of any other stack. Hence, without loss of generality, we assume that container c is not yet placed at the bottom of a stack.

We distinguish two cases: (i) the situation in which the origin and destination stacks s and s' are different, and (ii) the situation in which they are the same. In the first case, we need to have available slots for all containers in the destination stack s' which are at most H containers. Moreover, at most $H-2$ containers are placed on top of container c. Hence, at most $H-2$ slots should be available for containers from the stack s. Therefore, the total number of free slots needed to move container c from the origin stack to the destination stack is $H+H-2=2(H-1)$.

In the second scenario, in which the origin and the destination stack are the same, the reasoning is slightly different. It is important to realize that we need to store container c temporarily in another stack and that no other containers from the origin stack might be placed on top of container c because otherwise it cannot be placed again in the origin stack. Hence, we need at least two stacks with available slots. Since container c should block the smallest number of empty slots as possible it is placed on the stack that already consists of the most containers. Let us denote the stack in which container c is temporarily stored by s'' and let $p(c)$ be the position of container c in stack s''. After container c is placed in stack s'' we cannot use stack s'' anymore for the other containers. The number of available slots that cannot be used in stack s'' equals $H-p(c)$. Hence, according to the condition of this lemma, there should be at least $2(H-1)-(H-p(c)) = H+p(c)-2$ slots available for the other containers in the origin stack. Since $p(c)$ is by definition at least 1, there are at least $H-1$ slots available for other containers and they can thus be placed in other stacks than stack s''. Afterward, container c can be placed at the bottom of the origin stack. □

Note that in Lemma 3 we do not impose the condition that there are more than two stacks because this lemma also hold if the number of stacks is one or two. If there is only one stack, all $2(H-1)$ free slots should be in that single stack. The inequality $2(H-1) \leq H$ only holds for the trivial bay in which $H=1$ and there is only one container, or $H=2$ and there are no containers. In case there are two stacks, there can be at most two containers in the bay if the number of free slots is at least $2(H-1)$. With two containers, both containers can be placed at the bottom of a stack, so the lemma also holds.

The bound given in Lemma 3 is tight. To see that, consider a bay with three stacks, in which one has a height H, the second has a height 2 and the third stack contains only a single container. The number of free slots in this bay equals $H-2+H-1=2(H-1)+1$. In order to place the top container of the second stack in the third stack, the third stack needs to be empty. However, there is no place for the container of the third stack in the first stack. Furthermore, it is also not possible to move the top container of the second stack at the bottom of the second stack. We could move the top container of the second stack to the third stack, but then the bottom container of the second stacks has no stack in which it can be placed.

The minimum number of available slots given by Lemma 3 can be used to get a bound on the maximum number of containers that can be in a bay such that every container can become well-placed in a stack without moving containers from a third stack. In Corollary 2, we derive a bound on this number of containers. The proof of this corollary is similar to the proof of Corollary 1.

Corollary 2. *If the number of containers C is bounded by $C \leq SH-2(H-1)$, then every container could be moved from stack $s \in \mathcal{S}$ in the pre-processing phase to stack $s' \in \mathcal{S}$ such that it is well-placed without moving containers from stacks $\mathcal{S} \setminus (\{s\} \cup \{s'\})$.*

4.3. Upper bound

It is not hard to find an upper bound for the optimal solution because any feasible set of pre-processing moves P and policy for relocation moves π is a feasible solution for the SCRPPP and thus an upper bound for the optimal solution. In case $P=\emptyset$, no pre-processing moves are performed and all moves will be done in the relocation phase. Hence, under this strategy the SCRPPP will get the same solution as the SCRP.

Another strategy could be to continue with the pre-processing phase as long as there are containers that are not correctly placed, which is equivalent to the CPMP. In the latter situation, one needs to make the assumption that the number of containers in a time interval is always less than the number of stacks. Otherwise, there will always be two containers with the same time interval being stacked on top of each other, which results in an expected number of relocation moves of at least $\frac{1}{2}$.

For large values of α, not performing any pre-processing moves is a rather good strategy, while for values of α close to 0 using the algorithms for the CPMP results in good solutions. An upper bound for the optimal solution of the SCRPPP is the minimum value of the objective function under these two strategies. This upper bound can be used as a benchmark to evaluate the benefits of the combination of a pre-processing and a relocation phase. Furthermore, it can also be used to get the maximum number of pre-processing moves in the optimal solution as we will see in Lemma 4.

Lemma 4. *The maximum number of pre-processing moves p in the optimal solution is bounded by $p \leq \left\lfloor \frac{UB}{\alpha} \right\rfloor$.*

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

This lemma follows directly from the fact that performing p pre-processing moves yields an objective function of at least $p\alpha$. We will need this maximum number of pre-processing moves for the branch and bound algorithm described in Section 5.3 below.

4.4. Lower bound

Besides a good upper bound, also a tight lower bound is needed for the branch-and-bound algorithm that will be discussed in Section 5.3. Finding a good lower bound for the SCRPPP is more complicated than the upper bound of the previous section. There are multiple lower bounds known for the (S)CRP, (see, e.g., Galle et al., 2018 and Scholl, Boywitz, & Boysen, 2018). One lower bound for the CRP is to count how many containers have a container with a lower time frame underneath them. These containers need to be relocated at least once and all other container will never be relocated. Hence, a lower bound for the CRP for bay B is:

$$LB_{CRP}(B) = \sum_{c \in \mathcal{C}} \mathbb{1}_{t(c)<u(c)}.$$

The lower bound for the SCRP from (Galle et al., 2018) is similar but also takes into account the fact that two containers in a stack might have the same time frame. In this lower bound, the probability that a container is blocking another container is calculated. If for a container c holds that $t(c) < u(c)$, then it will never be relocated and if $t(c) > u(c)$ the probability of a relocation move is 1. If $t(c) = u(c)$, it is slightly more complex to calculate the desired probability. To derive that probability, let us define $m(c)$ as follows:

$$m(c) := |\{c' \in \mathcal{C} : s(c) = s(c') \wedge p(c) > p(c') \wedge u(c) = t(c) = t(c')\}|. \quad (2)$$

The number $m(c)$ represents the number of containers that are located underneath container c and that have the same time frame. The only possibility that container c does not have to be relocated, is when it is retrieved before any of the $m(c)$ containers. This scenario occurs with probability $\frac{1}{m(c)+1}$. Hence, the probability that container c does need to be relocated is $1 - \frac{1}{m(c)+1} = \frac{m(c)}{m(c)+1}$. Combining this expression with the lower bound for the CRP the lower bound for the SCRP of Galle et al. (2018), $LB_{SCRP}(B)$, is formally given by:

$$LB_{SCRP}(B) = \sum_{c \in \mathcal{C}} \mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}. \quad (3)$$

From this lower bound for the SCRP, we can derive a trivial lower bound for the SCRPPP. The container c needs to be moved at least $\mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}$ times. Ideally, this move is done in the pre-processing phase at a cost of α, thus a lower bound for the SCRPPP is $\alpha LB_{SCRP}(B)$.

The bay in Fig. 4 will be used to illustrate the lower bound for the SCRPPP. In this bay, all but two containers have only containers with a larger time frame underneath them. The two containers that have a positive number of relocation moves are the containers with time frame 3 in the first stack and the container with time frame 2 in stack 1 that is located on top of the other container with time frame 2. The probability that the container with time frame 3 is relocated is 1 and the probability that the container with time frame 2 is relocated is $\frac{1}{2}$. Therefore, the lower bound of Eq. (3) for this bay is $1\frac{1}{2}$, so the trivial lower bound for the SCRPPP would be $1\frac{1}{2}\alpha$.

One extra insight helps us to get a tighter lower bound than $\alpha LB_{SCRP}(B)$: it is only possible to move a container in the pre-processing phase if all the containers on top of it are also moved in the pre-processing phase. If only a single container c is considered, the lower bound for the number of moves for that container

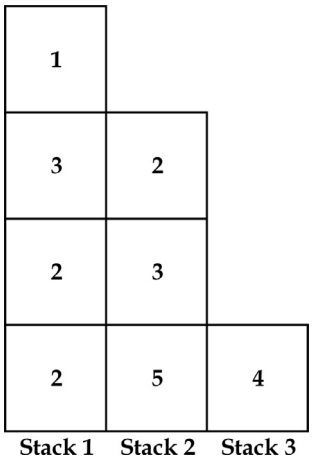


Fig. 4. Example of a bay layout to illustrate the lower bound.

c is given by:

$$\min\left\{\mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}, \alpha(n(s(c)) - p(c) + 1)\right\}. \quad (4)$$

The first part of the minimum is the expected number of relocation moves if the container is not moved in the pre-processing phase and is the same as in Eq. (3). The second part of the minimum is the cost associated with moving the container in the pre-processing phase. The height of the stack in which container c is positioned is given by $n(s(c))$. As $p(c)$ indicates the position of container c in stack $s(c)$, in total $n(s(c)) - p(c) + 1$ containers are moved if container c is moved in the pre-processing phase. It is not possible to sum the expression in Eq. (4) over all containers because if there are multiple containers in a stack for which $\alpha(n(s(c)) - p(c) + 1) \leq \mathbb{1}_{t(c)<u(c)} + \frac{m(c)}{m(c)+1}$ the pre-processing move of the top container of a stack is counted twice. Nevertheless, if a stack is considered from the top to the bottom, Eq. (4) can be used to obtain a lower bound for an entire bay. In Algorithm 1, it is shown in detail how this lower bound is constructed.

Algorithm 1 consists of two main for-loops. In the first for-loop, for each container the probability that it is relocated is calculated in the same way as in Eq. (3). In the second for-loop, for every container the current lower bound for that container and all of the containers on top of it is calculated. If the sum of these lower bounds is larger than α times the number of containers under consideration, pre-processing all these containers is beneficial. Hence, for each of these containers the lower bound is set to α. Otherwise, it is not beneficial to perform any pre-processing moves and the lower bound remains the same.

As an example, consider again the bay as given in Fig. 4. In order to move any container from the first stack, the container with time frame 1 on top of that stack also needs to be relocated. Thus, if we would like to relocate the container with time frame 3 from that stack in the pre-processing phase, at least two pre-processing moves are needed. Since these two pre-processing moves are only resulting in one relocation moves less, it is only beneficial if $\alpha \leq \frac{1}{2}$. So the lower bound produced by Algorithm 1 for the bay from Fig. 4 is 3α if $\alpha \leq \frac{1}{2}$ and $1\frac{1}{2}$ if $\alpha \geq \frac{1}{2}$. We will close this section with Lemma 5 that shows that the lower bound that is given by Algorithm 1 is tight.

Lemma 5. *The lower bound for the optimal solution of the SCRPPP given by Algorithm 1 is tight.*

Proof. The example in Fig. 4 can be used to prove this lemma. As stated previously, the lower bound for $0 < \alpha \leq \frac{1}{2}$ for this instance is 3α and for $\frac{1}{2} \leq \alpha < 1$ it is $1\frac{1}{2}$. It can be easily verified

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

Algorithm 1: Lower bound for the SCRPPP.

Input: Bay B and $0 < \alpha < 1$.
for $c \in \mathcal{C}$ **do**
  if $t(c) > u(c)$ **then**
    $lb(c) = 1$.
  else
    if $t(c) = u(c)$ **then**
      $lb(c) = \frac{m(c)}{m(c)+1}$.
    else
      $lb(c) = 0$.
    end
  end
end
for $s \in \mathcal{S}$ **do**
  for $t = 0 : n(s) - 1$ **do**
    $g_1(t) := \sum_{c:s(c)=s \wedge p(c) \geq n(s)-t} lb(c)$.
    $g_2(t) := (n(s) - t + 1)\alpha$.
    if $g_2(t) < g_1(t)$ **then**
      for $c \in \mathcal{C} : s(c) = s \wedge p(c) \geq n(s) - t$ **do**
        $lb(c) = \alpha$.
      end
    end
  end
end
Output: $LB = \sum_{c \in \mathcal{C}} lb(c)$.

that the expected number of relocation moves for the bay in Fig. 4 is $1\frac{1}{2}$, thus if one decides to perform no pre-processing moves, then the objective function for the SCRPPP is also $1\frac{1}{2}$. Another solution could be to perform the following pre-processing moves: $P = \{(1,2),(1,3),(1,3)\}$. After these three pre-processing moves, no relocation moves are needed and the value of the objective function of the SCRPPP is 3α. Hence, if we apply the first strategy for $\frac{1}{2} \leq \alpha < 1$ and the second strategy for values of $0 < \alpha < \frac{1}{2}$, the objective function for the SCRPPP is the same as the lower bound. □

5. Solution methods

In this section, we present both a heuristic method and an optimal branch-and-bound algorithm for the SCRPPP. As shown in Section 4.1, the SCRPPP is NP-hard, thus we expect that for larger instances the branch-and-bound algorithm cannot produce an optimal solution in reasonable time. Therefore, the local search heuristic is needed to produce solutions for large-sized instances. Moreover, the branch-and-bound algorithm needs a good upper bound for the optimal solution in order to run efficiently and the solution of the heuristic is a good upper bound for the optimal solution.

In Section 5.1, we will describe a local search heuristic to solve the SCRPPP. This heuristic needs, as a subroutine, a fast method to get an estimate for the expected number of relocation moves for a given bay. In Section 5.2, we will describe the method that is used to get such an estimation. Finally, an optimal algorithm for the SCRPPP is given in Section 5.3.

5.1. Local search heuristic

In this section, we will describe a local search method to solve the SCRPPP. Our main focus is on the pre-processing phase because in Galle et al. (2018) it is shown that the Expected Minmax (EM) heuristic produces fast and good solutions for the SCRP. Therefore, we will use that heuristic for the relocation phase. In Section 5.1.1, we first give a general overview of the local search heuristic. The heuristic is described in more detail in Section 5.1.2.

5.1.1. General overview

The general idea of the local search heuristic is to check for every container that is not correctly placed if there is a stack to move that container to such that the container is correctly placed and the objective function improves. We call this container the *investigated container*. Similar to the LPF heuristic for the CPMP, we start with investigating the movement of the containers with the highest time frame. These containers are the most difficult containers to place correctly because they can only be placed at the bottom of a stack. Let us call the stack to which we try to move the investigated container, the *destination stack*. We try every stack as possible destination stack and if there is a stack that yields an improvement, then the investigated container is moved to the stack that gives the largest improvement. If for none of the stacks the movement of the investigated containers gives an improvement in the objective function, then the next container is considered.

For the investigated container to be placed correctly in the destination stack, two types of containers need to be moved to other stacks. First of all, the investigated container might not be at the top of a stack, so the containers above the investigated container need to be moved to other stacks. Furthermore, to place the investigated container correctly in the destination stack, it might be needed to remove containers from that destination stack. Ideally, we would like to place both types of containers in a stack in which all containers are correctly placed and in which itself is also correctly placed. In that case, it is likely that we do not need to move the container a second time. If such a stack does not exist, the container is placed in a stack in which the minimum time frame is as low as possible. That decision is based on the fact that in this stack the fewest containers can be correctly placed.

5.1.2. Detailed description

In this section, we give a more detailed description of the local search heuristic. The structure of the local search is given in Fig. 5. Algorithm 2 is the main algorithm and in this algorithm, it is decided which container is the investigated container and to

Algorithm 2: Local search heuristic for pre-processing phase of the SCRPPP.

Input: Bay B and $0 < \alpha < 1$.
Initialize pre-processing moves $P = \emptyset$.
for $p = Z, Z-1, \ldots, 1$ **do**
  $A_p = \{c \in \mathcal{C} : t(c) = p\}$
  for $i = 1, \ldots, |A_p|$ **do**
    Select randomly a container c with time frame p that is not placed correctly,
    Use Algorithm 5 to get $f(B)$.
    for $s \in \mathcal{S}$ **do**
      Move container c to stack s according to Algorithms 3 and 4.
      Let $m(s)$ be the number of performed pre-processing moves and B_s the resulting bay.
      Use Algorithm 5 to estimate $f(B_s)$.
      $I(s) = f(B) - (\alpha m(s) + f(B_s))$.
    end
    if $\max_{s \in \mathcal{S}}\{I(s)\} > 0$ **then**
      Place container c in stack $s' = \arg\max_{s \in \mathcal{S}}\{I(s)\}$.
      Add the respectively pre-processing moves to PThe resulting bay is the new bay B.
    end
    $A_p = A_p \setminus \{c\}$.
  end
end
Output: Set of pre-processing moves P.

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

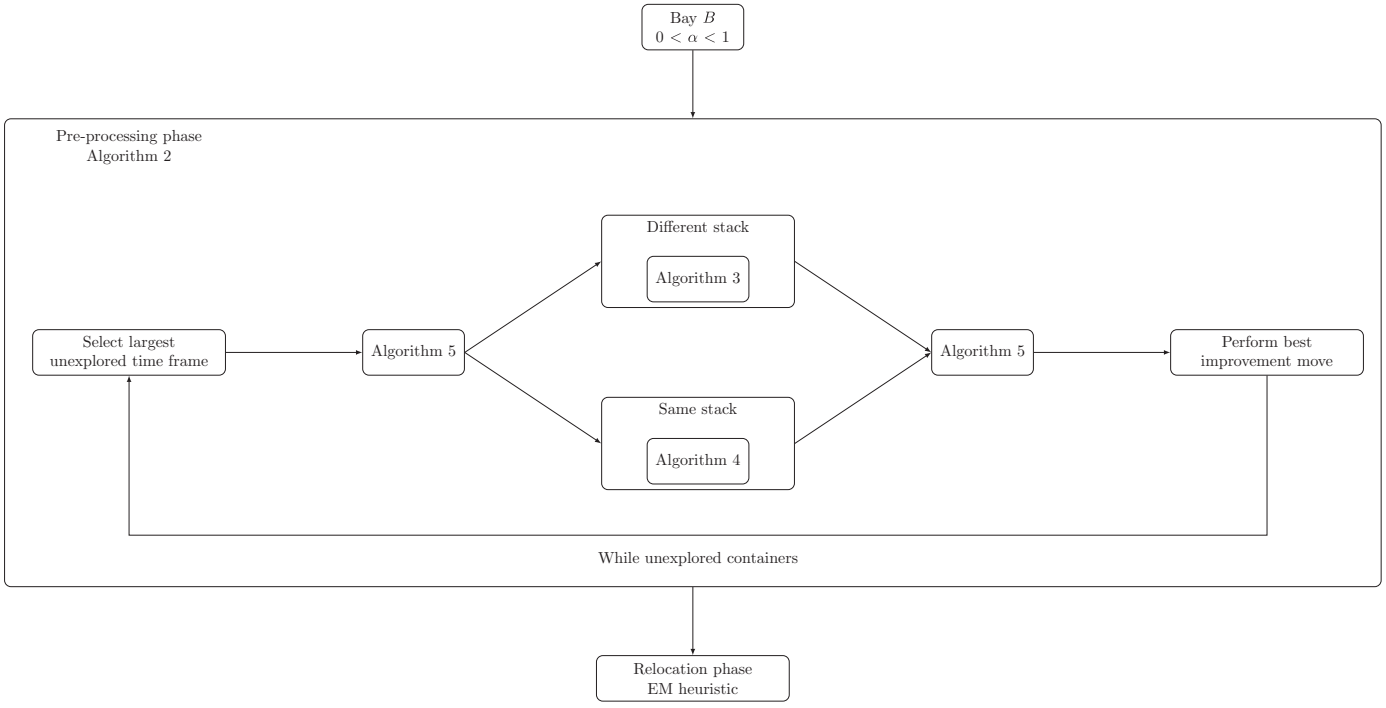


Fig. 5. Structure of the local search heuristic for the SCRPPP.

which stack it should be moved. In Algorithms 3 and 4 , it is decided how all containers should be moved such that the investigated container can be placed in the destination stack. The difference between Algorithms 3 and 4 is that Algorithm 3 is used if the destination stack is different from the stack in which the investigated container is located and Algorithm 4 if these two stacks are the same.

In Algorithm 2, the containers are considered in decreasing order of their time frame. If there are multiple containers with the same time frame, a random container is selected. We use Algorithm 5 to get an estimate for the number of relocation moves in a bay ($f(B)$). We use Algorithm 3 and 4 to move the container to another stack. After that, Algorithm 5 is run again to get an estimate for the number of relocation moves in the new bay and with that the improvement in the objective function ($I(s)$) can be calculated. If the number of containers in a bay is fewer than or equal to $SH - 2(H-1)$, we know by Corollary 2 that every container can be placed in every stack using Algorithms 3 and 4. In case there are more than $SH - 2(H-1)$ containers in the bay, it might be infeasible to place a container in a certain stack. If a move is infeasible, the improvement is set to $-\infty$. If there is a stack that gives a strictly positive improvement, the container is moved to the stack that gives the best improvement.

The way that a container is moved to a given stack is described in Algorithms 3 and 4. Algorithm 3 is used to move a container to a different stack than that it is currently located, whereas Algorithm 4 is used when the container needs to be positioned in a correct position in its current stack. In Algorithm 3, we need to move three different types of containers. The first type is container c itself that needs to move to its destination stack. However, it is only possible to access container c if we have moved the set of containers that are stacked on top of container c. We denote this second set by O_c. Finally, container c needs to be positioned in stack s such that it does not need to be relocated. All containers that need to be removed such that container c can be placed correctly in stack s are in the set M_c. Before container c can be moved, the containers in O_c and M_c need to be moved to a different

Algorithm 3: Pre-processing moves to move container c to a correct position in stack $s \neq s(c)$.

Input: Bay B with S stacks, container c, stack s, $0 < \alpha < 1$, $1 \leq \lambda_1 \leq S$ and $1 \leq \lambda_2 \leq S$.
$P = \emptyset$.
$O_c = \{c' \in \mathcal{C} : s(c') = s(c) \wedge p(c') > p(c)\}$.
$M_c = \{c' \in \mathcal{C} : s(c') = s \wedge u(c') < t(c)\}$.
while $O_c \cup M_c \neq \emptyset$ **do**
  Let o and m be the top container of, respectively, the sets O_c and M_c.
  if $t(m) \geq t(o)$ **then**
    $c' = m$.
    $M_c = M_c \setminus \{c'\}$.
    $p_1 = s$.
  else
    $c' = o$.
    $O_c = O_c \setminus \{c'\}$.
    $p_1 = s(c)$.
  end
  if *There exists a stack* $s' \in \mathcal{S} \setminus (\{s(c)\} \cup \{s\})$ *for which* $n(s') < H$ *and* $l(s') > t(c)$ *and* $f(B, s') \leq \alpha$ **then**
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\}$ with $n(s') < H$ and $f(B, s') \leq \alpha$, select randomly a stack s'' out of the at most λ_1 stacks with the smallest values for $l(s') > t$.
  else
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\})$ with $n(s') < H$, select randomly a stack s'' out of the at most λ_2 stacks with the smallest values for $l(s')$.
  end
  Relocate container c' to stack s''.
  $P = P \cup \{(p_1, s'')\}$.
end
$P = P \cup \{(s(c), s)\}$.
Output: Pre-processing moves P and bay B.

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

Algorithm 4: Pre-processing moves to move container c to a correct position in stack $s = s(c)$.

Input: Bay B with S stacks, container c, stack s, $0 < \alpha < 1$, $1 \leq \lambda_1 \leq S$ and $1 \leq \lambda_2 \leq S$.
$P = \emptyset$.
$S_1 = \{c' \in \mathcal{C} : s(c') = s \wedge p(c') > p(c)\}$
$S_2 = \{c' \in \mathcal{C} : s(c') = s \wedge p(c') < p(c) \wedge u(c) < t(c)\}$
while $S_1 \neq \emptyset$ **do**
  Let c' be the top container of stack s. **if** *There exists a stack* $s' \in \mathcal{S} \setminus \{s(c)\}$ *for which* $n(s') < H$ *and* $l(s') > t(c')$ *and* $f(s') \leq \alpha$ **then**
    For all stacks $s' \in \mathcal{S} \setminus \{s\}$ with $n(s') < H$ and $f(B, s') \leq \alpha$, select randomly a stack s'' out of the the at most λ_1 stacks with the smallest values for $l(s') > t(c')$.
  else
    For all stacks $s' \in \mathcal{S} \setminus s$ with $n(s') < H$, select randomly a stack s'' out of the the at most λ_2 stacks with the smallest values for $l(s')$.
  end
  Relocate container c' to stack s''.
  $S_1 = S_1 \setminus \{c'\}$.
  $P = P \cup \{(s, s'')\}$.
end
Move container c to the highest stack s''' with $n(s''') < H$ and update $s(c)$
$P = P \cup \{(s, s''')\}$.
while $S_2 \neq \emptyset$ **do**
  Let c' be the top container of stack s.
  if *There exists a stack* $s' \in \mathcal{S} \setminus (\{s(c)\} \cup \{s\})$ *for which* $l(s') > t(c')$ *and* $n(s') < H$ *and* $f(s') \leq \alpha$ **then**
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\})$ with $n(s') < H$ and $f(B, s') \leq \alpha$, select randomly a stack s'' out of the at most λ_1 stacks with the smallest values for $l(s') > t(c')$.
  else
    For all stacks $s' \in \mathcal{S} \setminus (\{s\} \cup \{s(c)\})$ with $n(s') < H$, select randomly a stack s'' out of the at most λ_2 stacks with the smallest values for $l(s')$.
  end
  Relocate container c' to stack s''.
  $S_2 = S_2 \setminus \{c'\}$.
  $P = P \cup \{(s, s'')\}$.
end
Move container c to stack s.
$P = P \cup \{(s(c), s)\}$.

Algorithm 5: Algorithm to estimate the expected number of relocation moves for a bay B..

Input: Bay B
for *All containers* c *in* $\mathcal{C}$ **do**
  if $u(c) > t(c)$ **then**
    $q(c) = 1$.
  else
    if $u(c) = t(c)$ **then**
      $q(c) = 2$.
    else
      if $u(c) > \min_{s \in \mathcal{S}}\{h(s)\}$ *or* $t(c) < \max_{s \in \mathcal{S}}\{l(s)\}$ **then**
        $q(c) = 3$.
      else
        $q(c) = 4$.
      end
    end
  end
end
Output: $\sum_{c \in \mathcal{C}} 1.4 * \mathbb{1}_{q(c)=4} + \mathbb{1}_{q(c)=3} + 0.5 * \mathbb{1}_{q(c)=2}$.

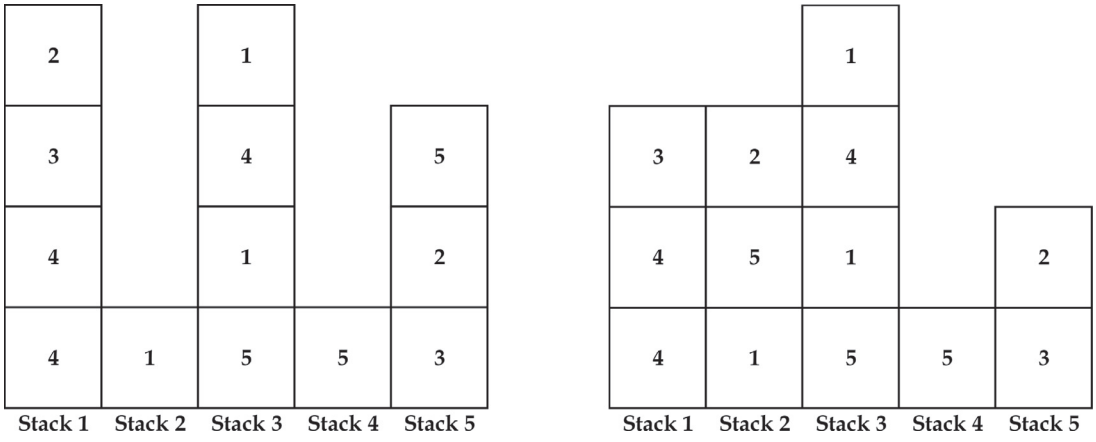
Algorithm 6: Branch-and-bound algorithm to solve the SCRPPP to optimality.

Input: Bay B and $0 < \alpha < 1$
$UB :=$ the solution of the pre-processing moves of Algorithm 2 with $\lambda_1 = \lambda_2 = 1$ and relocation moves of the PBFS algorithm.
$LB :=$ the lower bound from Algorithm 1.
$d := \lfloor \frac{UB}{\alpha} \rfloor$
$OPT := UB$
$Q := (B, d, LB)$
$I := \emptyset$
while $LB < OPT$ *and* $Q \neq \emptyset$ **do**
  Find the triplet $(B', d', LB') \in Q$ with the smallest LB'. If there are multiple bays, choose the bay with the smallest value of d'. In case there are still multiple bays left, choose the bay that was the earliest added to Q.
  Compute the optimal expected number of relocation moves R for bay B' using the PBFS algorithm.
  $Q = Q \setminus \{(B', d', LB')\}$ and $I = I \cup \{(B', d', LB')\}$
  Set $VAL = R + \alpha(d - d')$.
  if $VAL < OPT$ **then**
    $OPT = VAL$
  end
  if $d' > 0$ **then**
    for $s_1, s_2 \in S$ *and* $s_1 \neq s_2$, $n(s_1) > 0$ *and* $n(s_2) < H$ **do**
      Compute the lower bound LB'' for bay $B'' = B'(s_1, s_2)$ using Algorithm 1.
      if $LB'' + (d - (d' - 1)) * \alpha < OPT$ *and* $\{(B, d, LB) \in Q \cup I : B = B'' \wedge d \geq d' - 1\} = \emptyset$. **then**
        $Q = Q \cup \{(B'', d' - 1, B'')\}$
      end
    end
  end
end
Output: OPT

stack. From the containers in the sets O_c and M_c only the top container can be relocated. In Algorithm 3, the container with the largest time frame of these two top containers is moved first. The container with the largest time frame is selected because if that container can be correctly placed, the other container can be correctly placed on top of it.

When a container is moved to a different stack, the algorithm checks first if there exists a stack that satisfies the following three conditions: the first condition is that the stack has an empty slot. Secondly, the stack should only contain containers with a higher time interval than the container we are going to place in that stack because then the container does not need to be relocated. The final condition is that no pre-processing moves are performed for containers in that stack. This is equivalent to checking whether the expected number of relocation moves for that stack is fewer than or equal to α. A stack that satisfies these three conditions is referred to as a *correct stack*. If there are multiple correct stacks, the stacks for which the smallest time frame is as small as possible are preferred. Since these stacks are for fewer containers correct stacks. If there are fewer than λ_1 correct stacks, a random correct stack is selected. In case there are at least λ_1 correct stacks, we pick randomly one of the λ_1 stacks with the smallest minimum time frame.

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067



(a) Situation after the container with time frame 5 from the fourth stack of Figure 2 is placed in the fourth stack according to Algorithm 3.

(b) Situation after the container with time frame 5 from the fourth stack of Figure 2 is placed in the fourth stack according to Algorithm 4.

Fig. 6. Two different outcomes after moving the container with time frame 5 in the fourth stack of the bay of Fig. 2 in the pre-processing phase if $\alpha \geq \frac{1}{2}$.

If there are no correct stacks for a container, we would like to place the container in a stack with the minimum smallest time frame. This approach is similar to the LPF heuristic and its rationale is that a stack that has a container with a low time frame is less likely to be a correct stack for other containers. Hence, it is better to fill this stack with a container that has to be moved again than a stack that has for more containers the potential to be a correct stack. We select randomly one of the at most λ_2 stacks with the smallest minimum time frames.

Algorithm 4 is similar to Algorithm 3 when it comes to deciding the destination stack of a container. However, the sets of containers that need to be moved are different. Besides container c, there are again two different sets: S_1 and S_2. Set S_1 contains all containers that are placed on top of container c and set S_2 consists of all containers that are placed below container c and need to be moved to another stack in order to place container c in a correct position. Contrary to Algorithm 3, we do not need to determine which of the two top containers from the sets need to be moved first because the containers in set S_2 can only be moved when the containers from the set S_1 are moved to other stacks. After all containers in the set S_1 are moved to other stacks, container c needs to be temporarily stored in another stack. The containers of set S_2 cannot be placed in that stack because then it is impossible to move container c back to stack s. As a consequence, in Algorithm 4 container c is placed in the highest stack because then the fewest slots for the containers in set S_2 are blocked.

Running example continued

We will illustrate Algorithms 2, 3, and 4 with the bay as given in Fig. 2. Moreover, we set the value of α to $\alpha \geq \frac{1}{2}$ and the parameters λ_1 and λ_2 are both set to 2. Algorithm 2 first examines the containers with the largest time frame, in this case time frame 5. Let us say that the container with time frame 5 in stack 4 is the first container to be moved in the pre-processing phase. Furthermore, let us denote this container by c. We will first discuss how this container is placed in the second stack using Algorithm 3, and then, how it is placed according to Algorithm 4 in the fourth stack. In Fig. 6(a), the situation after the placement of container c in the second stack is depicted and the bay after that container c is moved to the fourth stack is shown in Fig. 6(b).

If container c is placed in stack 2, then set O_c is the container with time frame 3 above container c and the set M_c is equal to the two containers located in the second stack. Since the time frame of container c is larger than 1, also the bottom container of the second stack needs to be removed. The order in which the containers from the sets O_c and M_c are moved to other stacks is: first the container with time frame 5 from M_c, second the container with time frame 3 from O_c and finally the container with time frame 1 from M_c. When the first container from M_c needs to be moved to another stack there is no correct stack for the container. Hence, it is placed in one of the $\lambda_2 = 2$ stacks with the smallest time frames that have an empty slot. Since stack 3 has no empty slot, there are only two stacks possible, namely 1 and 5. Let us say that the container is placed in the fifth stack. Thereafter, the container with time frame 3 from the set O_c is moved. The expected number of relocation moves for the first stack and because $\alpha \geq \frac{1}{2}$, the first stack is a correct stack for this container. Moreover, it is the only correct stack for this container, thus the container is always placed in that stack. Finally, the container with time frame 1 from M_c has stacks 1 and 5 as correct stacks. Let us assume that it is placed in the first stack, then the situation after container c is placed in the second stack is as given in Fig. 6(a).

When container c is placed in the fourth stack, the set S_1 is the container with time frame 3 above container c and the container with time frame 2 constitutes on its own the set S_2. Similar to the situation in Fig. 6(a), the first stack is the only correct stack for the container with time frame 3. Afterward, container c is also placed in the first stack because that is the only stack that has a height of 3. Thereafter, no correct stack exists for the container with time frame 2 of set S_2. It could be placed in both stack 2 and 5 because stacks 1 and 3 have already reached its maximum height. In Fig. 6(b), we have assumed that the container is placed in the second stack after which container c could be placed back to the fourth stack.

In both Figs. 6(a) and (b) four pre-processing moves have been performed from the situation as shown in Fig. 2. The final bays for both scenarios have in common that container c is placed at the

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

bottom of a stack and thus that it does not need to be relocated. However, to examine which situation is better, we need to know the expected number of relocation moves for the complete bay. A method how this number can be estimated could be done is discussed in the next section.

5.2. Estimation number of relocation moves

For the local search heuristic described in Section 5.1, a fast estimation method for a bay's number of expected relocation moves is needed. It is important to note that in the heuristic, as sketched in Fig. 5, the relocation phase will not be solved to optimality, but instead the EM heuristic is used. Hence, we are not interested in an estimation of the optimal number of relocation moves, but the number of moves for the EM heuristic. One way to obtain the expected number of relocation moves for a bay is to simulate a set of retrieval orders and use the EM heuristic to compute the average number of relocation moves. However, in the local search heuristic of Algorithm 2, we need to compute the expected number of relocation for many different bays, so a faster method is preferred. Moreover, simulation will inherently result in stochastic outcomes and we prefer to have a deterministic estimation. Therefore, we use a rule-based estimation method that is given in Algorithm 5.

The main idea of Algorithm 5 is to estimate the number of relocation moves needed for one specific container. If a container has only containers with a higher time frame underneath it, it is obvious that it will never be relocated. However, if a container needs to be relocated, it is hard to get the exact number of relocation moves that will be needed for that container. In order to get a good estimation of the number of relocation moves we divide the containers into four categories: category 1, 2, 3 and 4. Containers in category 1 do not need to be relocated. For the containers in categories 2, 3 and 4 we expect less than one, exactly one and more than one relocation move, respectively. It is important to note that we use the word 'expect' here because it is hard to compute exactly how often a container needs to be relocated.

In Algorithm 5, the containers are divided into different categories and these categories are used to get an estimation for the expected number of relocation moves. The algorithm iterates through all containers in a bay and consists of three if-statements. In the first if-statement, it is checked if the smallest time frame of the containers underneath container c is higher than the time frame of container c. If this is the case, container c will definitely not be relocated and it is assigned to category 1. Secondly, if the smallest time frame of the containers underneath c is exactly the same as the time frame of container c, container c is assigned to category 2. In this situation, it is unsure if container c will need to be relocated. If it is to be retrieved before the container(s) with the same time frame in the same stack, it does not need to be relocated. However, it might also happen that container c is retrieved after a container from the same stack and the same interval. In that case, container c will have to be relocated to another stack. Nevertheless, at that time all containers with a lower time frame have already left the bay, so container c can, most likely, be placed in a stack in which it does not have to be relocated again. All in all, the expected number of relocation moves for the containers in category 2 is likely to be less than one.

After the first two if-statements, all containers that are not yet assigned to a category have at least one container with a lower time frame underneath them. These containers will always be relocated at least once. In the third if-statement, it is checked whether it is possible that there is a good stack available for container c at the first time it will be relocated. A good stack is defined as a stack in which container c will not need to be relocated a second time. It is hard to determine exactly, in an efficient way, whether there exists a good stack for a container at the time it is relocated. This difficulty lies in the fact that we do not know the layout of the bay before the container under consideration is relocated. Hence, we use two rules of thumb to check whether it is likely that a good stack is available for container c at the time it needs to be relocated.

The first rule of thumb exploits the fact that container c will be relocated at time $u(c)$. If there is a stack for which all containers have a lower time frame than $u(c)$, all containers from that stack will already be retrieved at time $u(c)$. Therefore, that stack might be empty at time $u(c)$ and container c could be placed in that stack without any further relocation moves. The second rule of thumb checks whether there exists a stack for which all time frames are higher than the time frame of container c. In case there exists such a stack, it is likely that container c can be placed on that stack without any further relocation moves. For both rules of thumb, it might be that the good stack is not available anymore at the time the container needs to be relocated because another container is already placed in that stack. If one of the two rules are satisfied, container c belongs to category 3, otherwise, it falls under category 4.

After all containers have been assigned to a category, the number of relocation moves for a bay is estimated using the formula:

$$\sum_{c \in \mathcal{C}} 1.4 * \mathbb{1}_{q(c)=4} + \mathbb{1}_{q(c)=3} + 0.5 * \mathbb{1}_{q(c)=2}. \tag{5}$$

The coefficients for the categories in this sum are both based on logical reasoning and numerical experiments. Below we will explain how these coefficients are derived. We definitely know that the containers in category 1 will never be relocated, so they get a weight of 0 in the sum. In order to find a good coefficient for the containers of categories 2, 3 and 4, we have used linear regression. The 'true' number of relocation moves is determined by simulating 1,000 retrieval orders for each of the 1,440 instances for the SCRP as created by Ku and Arthanhari (2016) and by using the EM heuristic to solve them. This average number of relocation moves for an instance will act as an independent variable. For each instance, the number of containers in categories 2, 3 and 4 are explanatory variables. Hence, there are 1,440 independent variables that could be explained by three dependent variables. If linear regression is used, we get the coefficients 0.515 for the category 2 containers, 1.035 for category 3 and 1.39 for category 4 and an R^2 of 0.968. If we round the coefficients for the categories 2, 3 and 4 to respectively 0.5, 1 and 1.4, then the R^2 only slightly decreases to 0.966. Therefore, we choose these values in Eq. (5) and Algorithm 5.

The mean absolute percentage difference between the outcome of the simulations and Algorithm 5 which is only 5.16%. It should be noted that this is based on solving the original SCRP instances of Ku and Artanhari (2016). Nevertheless, it is easier to make a prediction for the expected number of relocation moves after some pre-processing moves have been performed. Since, the more pre-processing moves are done, the more containers are placed such that they do not need to be relocated. These containers belong to category 1 which is the only category for which we can derive the exact expected number of relocation moves. Hence, the further the local search heuristic proceeds the more accurate the prediction using Algorithm 5 will be.

Running example continued

To illustrate Algorithm 5, the containers as previously given in Fig. 2 are assigned to categories in Fig. 7. The green containers are all containers from the first category. All green containers either have no containers at all, or only containers with a strictly higher time frame underneath them. The containers in the second category are colored yellow. For these containers, the container with the smallest time frame underneath them has exactly the same

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

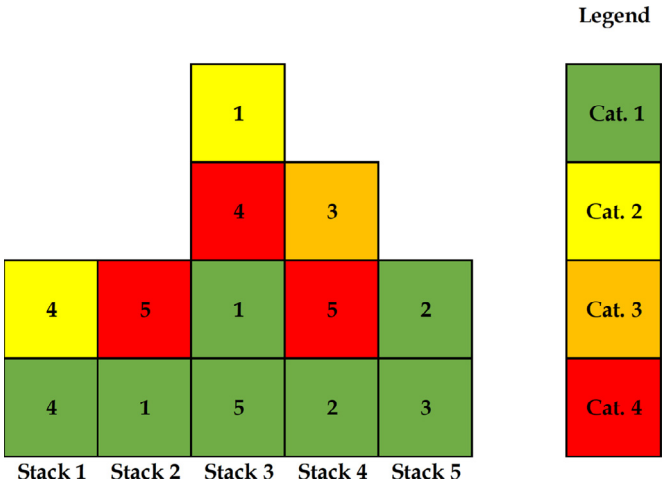


Fig. 7. Categories of containers in the bay from Fig. 2. The colors green, yellow, orange and red correspond to, respectively, to category 1, 2, 3 and 4.

time frame as they have. The top container of the fourth stack belongs to the third category and is thus orange because the second criterion of the third if-statement is true. The maximum of the smallest time frame of the first stack is 4, which is strictly smaller than 3. Hence, we expect that this container can be relocated to the first stack and afterward does not need to be relocated again. The red containers are from category 4 because they do not satisfy any of the conditions in the if-statements.

If Algorithm 5 is run for the bay of Fig. 7, we get an estimation of the total number of relocation moves that is equal to: $3 * 1.4 + 1 + 2 * 0.5 = 6.2$. Since this is a small example we can calculate the expected number of relocation moves used by the EM exactly, which is $6\frac{1}{3}$. All in all, the prediction of Algorithm 5 is rather accurate for this bay.

After a method to estimate the number of relocation moves is developed, we can look again at the two bays in Fig. 6 to see if these pre-processing moves yield an improvement to the original bay. In the bay in Fig. 6(a), there are two containers from category 2, namely the upper container with time frame 4 in the first stack and the upper container with time frame 1 in the third stack. Furthermore, two containers belong to category 3: the container with time frame 4 in the third stack and the container with time frame 5 in stack 5. The container with time frame 4 in the third stack belongs to category 2, because in stack 4 are only containers with a larger time frame. On the other hand, the container with time frame 5 in the fifth stack belongs to category 3 because stack 2 only contains a container with time frame 1, which is already retrieved before the container with time frame 5 needs to be relocated. Since, there are no containers from category 4 in the bay of Fig. 6(a), the number of relocation moves can be estimated by: $2 * 1 + 2 * 0.5 = 3$.

In the bay of Fig. 6(b), the same containers are in category 2 as in Fig. 6(a). Besides that, the container with time frame 4 in the third stack is also in category 3 together with the container with time frame 2 in stack 2. Both of these containers would namely be well-placed if we would move them to stack 4. The container with time frame 5 in the second stack, however, belongs to category 4. Therefore, estimated number of relocation moves for the bay of Fig. 6(b) is: $1 * 1.4 + 2 * 1 + 2 * 0.5 = 4.4$.

Recall that both the bays in Figs. 6(a) and (b) were obtained after four pre-processing moves from Fig. 2. Combined with the estimated number of pre-processing moves, the estimated objective of the bay in Fig. 6(a) is $4\alpha + 3$ and for the bay in Fig. 6(b) the estimated objective function is $4\alpha + 4.4$. The bay in Fig. 6 is thus always preferred over the bay in Fig. 6(b). Moreover, the four pre-processing moves that lead to this bay are only carried out in Algorithm 2 if $6.2 - (4\alpha + 4.4) > 0$, or stated otherwise if $\alpha < 0.45$.

5.3. Branch-and-bound algorithm

In this section, a branch-and-bound algorithm is presented to solve the SCRPPP to optimality. Similar to the local search heuristic in Section 5.1, we focus on the pre-processing phase because in Galle et al. (2018), an optimal algorithm for the relocation phase is given. The idea of the branch-and-bound algorithm is to investigate all possible pre-processing moves in the most efficient way. For this optimal algorithm, Lemma 4 (see Section 4.3) is crucial because it states that the maximum number of pre-processing moves in the optimal solution is $d := \lfloor \frac{UB}{\alpha} \rfloor$. On top of that, for a given bay, we could perform at most $S(S-1)$ different pre-processing moves because for every stack $s \in \mathcal{S}$ we could place its top container in every other stack. Therefore, in total there are at most $\sum_{i=0}^{d} (S(S-1))^i$ different solutions for the pre-processing phase of the SCRPPP. Even for small instances with 5 stacks, the number of possible solutions of the pre-processing phase is for $d = 7$ more than a billion.

In order to find the optimal number of relocation moves, the optimal algorithm for the relocation phase has to be applied to every single solution of the pre-processing phase. Since the relocation phase is NP-hard, the optimal algorithm of Galle et al. (2018) for the relocation phase does not run in polynomial time. Fortunately, using the branch-and-bound algorithm in Algorithm 6 it is possible to reduce the number of solutions of the pre-processing phase for which we need to find the optimal number of relocation moves.

In Algorithm 6, the branch-and-bound algorithm to solve the SCRPPP to optimality is given. The first step of this algorithm is to compute an upper bound for the optimal solution. Firstly, we use the local search heuristic of Algorithm 2 with $\lambda_1 = \lambda_2 = 1$ for the pre-processing phase. Afterward, we apply the Pruning-Best-First-Search (PBFS) algorithm of Galle et al. (2018) to obtain the optimal expected relocation moves for the resulting bay. The combination of the two algorithms will give a feasible solution for the SCRPPP and thus an upper bound of the optimal solution. Furthermore, Algorithm 1 is used to compute a lower bound for the optimal solution. In case the lower and upper bounds are the same, we have proven that the pre-processing moves of Algorithm 2 with $\lambda_1 = \lambda_2 = 1$ result in an optimal solution. Otherwise, we start with exploring all possible pre-processing moves in the while-loop in Algorithm 6. In this while-loop, we have a set of candidate solutions Q, and a set I of all solutions for which the optimal solution already has been computed. The while-loop ends either if we have found a set of pre-processing moves for which the objective function equals the lower bound of the initial bay or if the set Q is empty. In the while-loop, we select the element in Q for which we have the smallest lower bound. In case of a tie, we select the solution for which the most pre-processing moves have been performed. If there are still multiple of those solutions, we select the element that was the first to be added to Q. We prefer an element whose lower bound is smaller because this solution is the solution that is the most likely to have the lowest objective function.

For the element in Q that we have selected, the optimal number of relocation moves is calculated using the PBFS algorithm and with that, the objective function for that element is computed. In case the objective function is lower than the best-known solution found so far, OPT is updated. Afterward, if the remaining number of pre-processing moves in the current solution is larger than zero, we investigate all possible pre-processing moves for the current solution. For every possible resulting bay, we firstly compute its lower bound. In case this lower bound is smaller than OPT we

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

Table 2
Number of instances solved to optimality using Algorithm 6 and their running times.

		α	0.25		0.5		0.75	
H	S	Fill rate	50%	67%	50%	67%	50%	67%
3	5	Solved	30/30	30/30	30/30	30/30	30/30	30/30
		Time (s)	0.03	3.1	0.03	0.9	0.02	0.5
	10	Solved	30/30	27/30	30/30	29/30	30/30	30/30
		Time (s)	1.6	265.1	1.0	90.3	0.1	3.3
4	5	Solved	30/30	28/30	30/30	30/30	30/30	30/30
		Time (s)	1.6	426.1	0.6	94.3	0.2	0.5
	10	Solved	22/30	3/30	25/30	7/30	29/30	11/30
		Time (s)	644.0	1702.2	106.7	828.8	182.8	152.6
5	5	Solved	27/30	4/30	29/30	4/30	30/30	13/30
		Time (s)	462.4	1174.4	82.9	495.5	45.4	552.2
	10	Solved	3/30	0/30	7/30	1/30	16/30	4/30
		Time (s)	1707.9	–	1593.8	518.9	303.5	2112.0
6	5	Solved	9/30	0/30	14/30	0/30	20/30	1/30
		Time (s)	1068.9	–	722.4	–	271.4	81.9
	10	Solved	0/30	0/30	2/30	0/30	6/30	0/30
		Time (s)	–	–	73.9	–	917.2	–

check if the same bay with fewer pre-processing moves is not already in either Q or I. If the element is not already in Q or I it needs to be investigated and is added to Q. In checking whether a bay is already in Q or I we make use of the fact that a specific bay is considered to be the same as a bay in which the same stacks are placed in a different order.

6. Numerical results

In this section, we will use numerical experiments to check the quality of the local search heuristic and the branch-and-bound algorithm presented in Section 5. Both these methods will be evaluated according to their solution quality and running time. We will use the SCRP instances introduced by Ku and Arthanhari (2016) as our problem instances for the SCRPPP. This set of instances consists of 1,440 instances with the number of stacks ranging from 5 to 10 and the maximum stack height from 3 to 6. For half of the instances, the number of containers is half of the available slots and the other half of the instances have a fill rate of $\frac{2}{3}$. For each specific combination of number of stacks, maximum stack height and fill rate there are 30 instances. All these instances satisfy the condition of Corollary 2 which is that the number of containers is smaller than $SH - 2(H-1)$. Therefore, the local search heuristic can be applied to all instances without checking whether a move is feasible.

The remainder of this section is organized as follows. First, we will investigate in Section 6.1 the maximum size of an instance that can be solved to optimality by the branch-and-bound algorithm. Second, in Section 6.2, we will compare the results of the local search heuristic with the optimal solution. Finally, in Section 6.3, the solutions for the SCRPPP will be compared with the solutions for the SCRP and CPMP in order to see what the benefits of pre-processing are.

6.1. Optimal solution

To test the efficiency of the branch-and-bound algorithm of Section 6, we solve the instances of Ku and Arthanhari (2016) for different parameter settings. As stated before, the minimum number of stacks for these instances is 5 and the maximum is 10. We investigate the optimal solution for both this minimum and maximum number of stacks. The stack height can range from 3 to 6 containers and the fill rate of the bay can either be 50 % or 67%. For each combination, there are 30 instances that are all solved for the following values of α: $\alpha = 0.25$, $\alpha = 0.5$ and $\alpha = 0.75$. For every single instance and value of α the running time is set to at most one hour.

Table 2 shows how many of the 30 instances were solved to optimality within the hour. Furthermore, the average running time of all instances that were solved to optimality is given. As one can see, for the instances with five stacks and a maximum stack height of three, all instances were solved and on average it took at most a few seconds. However, for instances with a maximum stack height of six, almost none of the instances could be solved to optimality within an hour. The running time of the branch-and-bound algorithm increases extremely fast as the stack size increases. It can be concluded from Table 2 that the branch-and-bound algorithm is able to solve small instances for the SCRPPP, but that for larger instances the running time is too large. However, it should also be noted that there is a large fluctuation in the running time for different instance of the same size. For example, nineteen of the instance with $H = 4$, $S = 10$, $\alpha = 0.75$ and a fill rate of 67% are not solved to optimality within an hour. Nevertheless, the eleven instances that are solved to optimality within an hour, have an average running time of only 152.6 seconds.

The number of stacks has a much smaller influence on the running time than the maximum height of a stack. The bays with $S = 10$ and $T = 3$ and the bays with $S = 5$ and $T = 6$ have the same number of containers in them, but the former can be solved much faster than the latter. This can be explained by the fact that the number of moves in a bay with lower stacks is smaller than in a bay with higher stacks. Hence, both the lower and the upper bound for the SCRPPP is stronger if the stacks are lower. In case the value of the upper bound is larger, the value for d is also higher. Since the number of possible pre-processing moves is given by $\sum_{i=0}^{d} (S(S-1))^i$, the value of d has a larger impact on the size of the solution space than the value of S.

A similar reasoning applies to instances with a fill rate of 67%. These instances need more moves than instances with the same number of stacks and maximum height, but a fill rate of 50%. Therefore, the upper bound and the value of d is larger for instances with a fill rate of 67%. As a result, the running time for instances with a higher fill rate is larger. For some values of S, H and α the running time that is given in Table 2 is sometimes lower for instances with a fill rate of 67% than for 50%, see for instance $S = 10$, $H = 6$ and $\alpha = 0.75$. However, for these instances the number of instances with a fill rate of 67% that are solved within an hour is less than the number of instances with a fill rate of 50% that are solved in less than an hour.

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

Table 3
Objective function under the optimal relocation policy for the bay obtained after pre-processing according to the branch-and-bound algorithm (B&B), the local search heuristic for $\lambda_1 = \lambda_2 = 1$ (DPP) and the local search heuristic with $\lambda_1 = \lambda_2 = 3$ (RPP).

		α	0.25		0.5		0.75	
H	S	Fill rate	50%	67%	50%	67%	50%	67%
3	5	B&B	0.642	1.250	1.222	2.328	1.503	2.836
		RPP+PBFS	0.725	1.356	1.222	2.439	1.508	2.916
		DPP+PBFS	0.750	1.434	1.222	2.469	1.511	2.942
	10	B&B	1.000	1.867	1.922	3.534	2.581	4.606
		RPP+PBFS	1.192	2.083	2.172	3.622	2.592	4.631
		DPP+PBFS	1.200	2.175	2.172	3.656	2.592	4.647

Furthermore, from Table 2 it can be concluded that the larger the value of α, the faster the branch-and-bound algorithm runs. An explanation for this result is that the maximum number of pre-processing moves that can be used in the optimal solution is bounded by $\lfloor \frac{UB}{\alpha} \rfloor$. The larger the value of α, the fewer pre-processing moves could be performed and thus the fewer solutions need to be investigated in Algorithm 6. Finally, it should also be noted that the larger the value of α the smaller the difference between the instances with a fill rate of 50% and 67%.

6.2. Local search heuristic

In the previous section, we have seen that the branch-and-bound algorithm is able to find the optimal solution for small instances, but that it is not able to solve larger instances within reasonable time. This should not be surprising because we have shown that the SCRPPP is NP-hard. In this section, we will investigate whether the quality of the local search heuristic is close to the optimal solution for small instances and whether the running time of the heuristic is acceptable for larger instances.

The quality of the pre-processing moves that are performed in Algorithm 2 is compared with the optimal pre-processing moves in Algorithm 6. In order to make sure that we only look at the effect of the pre-processing moves, for both scenarios the relocation moves are solved to optimality according to the PBFS. We use two different variants of the local search heuristic. In the first variant, $\lambda_1 = \lambda_2 = 3$ is used and the second method uses $\lambda_1 = \lambda_2 = 1$. If $\lambda_1 = \lambda_2 = 3$, the local search heuristic is a randomized algorithm and we refer to it as *Randomized Pre-Processing* (RPP). Since it is a randomized algorithm, the algorithm is run 150 times or it stops after 100 runs without an improvement. This set-up is the same as the set-up of the LPFH in order to make a fair comparison possible in Section 6.3. After each of these runs, the optimal expected number of relocation moves for the bay after pre-processing is calculated. If Algorithm 2 uses $\lambda_1 = \lambda_2 = 1$ as parameters, it is a deterministic algorithm and we refer to it as *Deterministic Pre-Processing* (DPP). This deterministic algorithm is obviously only run once.

In Table 3, we give the objective function for the three methods described above for instances with five and ten stacks and with a maximum height of three. The value of α is set to 0.25, 0.5 and 0.75. The first thing to note is that the RPP is performing better than the DPP, but that the difference between them is quite small. Another observation is that the smaller the value of α, the larger the difference between the heuristic values and the objective obtained by the optimal branch-and-bound. This makes sense because for larger values of α fewer pre-processing moves are performed in all solutions and the relocation moves contribute for a large share to the objective function. Furthermore, the difference between the optimal and heuristic solution is larger for instances with ten stacks than with five stacks. Instances with ten stacks have more possible pre-processing moves, so it makes sense that the heuristic is performing worse for these.

Although the gap between the heuristic and the optimal solution differs per parameter setting, the average gap between the optimal branch-and-bound algorithm and the RPP + PBFS and DPP + PBFS over all instances reported in Table 3 is respectively 4.6% and 5.8%. Hence, we may conclude that the quality of the heuristic is reasonably good.

After it is shown that the value of the objective function of the DPP and RPP is close to optimal for small instances, we will solve larger instances with the DPP and RPP. In Galle et al. (2018), it is shown that the PBFS is not able to solve larger instances of the SCRP to optimality within an hour. Hence, we decided to use the EM heuristic for the relocation phase. We have chosen to use 1,000 simulations of the retrieval order to get the average number of relocation moves. In Table 4, the value of objective function for both the DPP + EM and RPP + EM is given. Furthermore, the average running time of the pre-processing phase and the relocation phase for the RPP + EM per instance is given. The deterministic DPP + EM is run only once, so the running times of the pre-processing phase and the relocation phase is an order of 100 smaller than for the RPP + EM. As the total average running time of the DPP + EM is always less than a second for every instance, we have chosen to only report the running times of the RPP in combination with EM

Table 4
Objective function for the DPP + EM and RPP + EM and the average running time per instance for the RPP + EM.

		α	0.25		0.5		0.75	
H		Fill rate	50%	67%	50%	67%	50%	67%
3	DPP+EM	Objective	0.98	1.79	1.72	3.15	2.08	3.93
	RPP+EM	Objective	0.96	1.72	1.70	3.11	2.08	3.89
		Time pre-processing (s)	0.2	0.4	0.1	0.4	0.1	0.4
		Time relocation (s)	0.2	0.2	0.1	0.7	1.0	4.2
4	DPP+EM	Objective	1.86	3.45	3.25	5.88	4.07	7.47
	RPP+EM	Objective	1.80	3.11	3.22	5.66	4.04	7.32
		Time pre-processing (s)	0.5	0.7	0.4	1.0	0.4	1.2
		Time relocation (s)	0.3	0.8	0.4	4.0	3.4	19.7
5	DPP+EM	Objective	3.14	5.65	5.46	9.81	6.83	12.55
	RPP+EM	Objective	2.88	4.93	5.29	9.16	6.74	11.88
		Time pre-processing (s)	0.9	1.6	0.9	1.9	1.1	2.7
		Time relocation (s)	0.8	1.8	3.6	10.2	14.3	42.5
6	DPP+EM	Objective	4.39	8.69	7.65	14.86	9.70	18.66
	RPP+EM	Objective	3.93	7.03	7.17	13.00	9.41	17.20
		Time pre-processing (s)	1.5	2.5	1.4	3.7	1.7	4.2
		Time relocation (s)	1.9	4.1	4.8	18.9	25.7	67.3

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

Table 5
Objective function for the RPP + EM, the EM and the LPFH.

	α	0.25		0.5		0.75	
H	Fill rate	50%	67%	50%	67%	50%	67%
3	RPP+EM	0.96	1.72	1.70	3.11	2.08	3.89
	EM	2.47	4.34	2.47	4.34	2.47	4.34
	LPFH	0.84	1.62	1.68	3.24	2.52	4.86
4	RPP+EM	1.80	3.11	3.22	5.66	4.04	7.32
	EM	4.63	8.12	4.63	8.12	4.63	8.12
	LPFH	1.63	3.16	3.26	6.33	4.89	9.49
5	RPP+EM	2.88	4.93	5.29	9.16	6.74	12.55
	EM	7.48	13.00	7.48	13.00	7.48	13.00
	LPFH	2.84	5.48	5.67	10.95	8.51	16.43
6	RPP+EM	3.93	7.03	7.17	13.00	9.41	17.20
	EM	10.32	18.43	10.32	18.43	10.32	18.43
	LPFH	4.03	9.55	8.07	19.10	12.10	28.65

heuristic in Table 4. Contrary to Tables 2 and 3, not only the results of the instances with five and ten stacks are given, but all instances with stacks five up to ten are used.

The total running time of the RPP + EM heuristic increases when the maximum stack size and α increase. Especially, the time needed for the EM heuristic in the relocation phase is larger when $\alpha = 0.75$. For this value of α, fewer pre-processing moves are performed and thus the relocation phase is harder. Although the running times of the RPP + EM are higher than for the DPP + EM, the objective for the RPP + EM is also smaller than for the DPP + EM. For $\alpha = 0.25$, the difference between the objective function for the two methods is the largest. This is caused by the fact that for this value of α more containers are moved in the pre-processing phase and for $\alpha = 0.75$ more containers are moved in the relocation phase. Furthermore, the more containers are in the bay the more one can gain by performing different pre-processing moves and thus the percentage difference between the RPP + EM and DPP + EM is bigger for larger values of H and a fill rate of 67%.

6.3. SCRP and CPMP

In this section, we will compare the newly proposed local search heuristic with two existing heuristics for the CPMP and SCRP. The LPF heuristic is a heuristic for the CPMP and can be used for the pre-processing phase. The LPF heuristic stops the moment no relocation moves are left. It is also possible to perform no relocation moves after which we could apply the EM heuristic for the SCRP for the relocation phase. The results of these two methods are compared with the RPP + EM method in Table 5.

The EM method does not use any pre-processing moves and thus the value of the objective function is independent of α. The LPF heuristic does perform pre-processing moves, but the number of moves is not influenced by α because it always tries to find the fewest moves such that no relocation moves are needed. Hence, the value of the objective function is just a linear function of α. We see in Table 5 that for α is 0.25 and 0.5, the LPF heuristic results in a lower value of the objective function than the EM heuristic. On the other hand for $\alpha = 0.75$ the EM heuristic gives a better solution.

The RPP + EM outperforms both the EM and LPFH for almost all instances. On average the objective function for RPP + EM is about 9% lower than that of the minimum of the EM and LPFH. Only for the instances with a small number of containers and $\alpha = 0.25$, the LPFH is slightly better. For $\alpha = 0.25$, it is beneficial to move a container in four pre-processing moves to a correct position. Hence, for smaller instances it is almost always possible to place all containers in the correct position. Since the LPFH is tailored for placing the containers in the correct position in the fewest moves, it is logical that it outperforms the RPP + EM.

Furthermore, we see in Table 5 that for large instances the difference between the RPP + EM and the EM and LPFH is bigger. This can be explained by the fact that the more containers are in a bay, the more difficult it is to place a container in the correct position in the pre-processing phase. Furthermore, if a container is moved in the relocation phase, then for larger instances it has more containers on top of it. Hence, for larger instance more pre-processing and relocation moves are needed and the more can be gained by balancing those two type of moves. Similarly, the trade-off between pre-processing and relocation moves is more important if $\alpha = 0.5$ and thus the difference between the RPP + EM and the EM and LPFH is also bigger for this value of α. For instance, for $\alpha = 0.5$, $H = 6$ and the fill rate of 67% the improvement is even more than 40%.

7. Conclusion

In this paper, we have introduced a new optimization problem that is faced by an inland container terminal in the port of Amsterdam: the Stochastic Container Relocation Problem with Pre-Processing. In this problem, the best trade-off has to be found between moving containers in the pre-processing phase and in the relocation phase. We have developed an optimal branch-and-bound algorithm to solve this problem. Since we have also proven that this problem is NP-hard, we have developed a local search heuristic to solve larger instances. That heuristic makes use of a newly developed estimation method for the number of relocation moves. For small instances, the heuristic gives close to optimal solutions. Furthermore, for larger instances and moderate values of α a large improvement is made compared to only moving containers in the pre-processing or relocation phase.

The optimal branch-and-bound algorithm that we propose could be improved if sharper upper and lower bounds are derived. If the upper bound is smaller, then the search tree could be pruned earlier. Furthermore, if the lower bound is larger, then the optimal number of relocation moves would need to be calculated for fewer bays. In the lower bound that we use, the moves needed to remove containers from a stack in order to place another container in that stack are not taken into account. It is not straightforward how one would incorporate these moves because if one stack is empty it might be that many containers benefit from that and could be placed in that stack. Nevertheless, taking these moves into account would improve the lower bound.

The local search heuristic that we have proposed might improve if a better estimation method for the number of relocation moves is used. The quality of this estimation method is important in our local search algorithm because it is used in deciding whether the bay after pre-processing moves has improved. The rule-based based method is effective but simple, and it might be that other estimation methods that, for instance, use machine learning will result in better prediction and thus an improvement of the performance of the local search heuristic. If the local search heuristic is improved, also a better upper bound for the branch-and-bound algorithm is available.

The current model is a first step in incorporating pre-processing in solving the SCRP and showing that a balanced trade-off between pre-marshalling and relocation moves is possible. However, in practice it might be more realistic to limit the total number of pre-processing moves that can be done. The branch-and-bound algorithm benefits from a limited number of pre-processing moves because then the search tree is smaller. However, although we could incorporate a maximum number of pre-processing moves in our current local search heuristic, it is not likely to give good results. The reason for this is that if the number of pre-processing

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067

moves is limited it might not be beneficial to start with performing the moves on the container with the largest time frame. It could be better to pre-process the containers that can easily be correctly placed. Hence, another solution method would be needed for this problem.

Acknowledgments

This work was partly supported by a Public-Private Partnership between the Centre for Mathematics and Computer Science (CWI) and container terminal CTVrede in the Netherlands.

References

Akyüz, M. H., & Lee, C. Y. (2014). A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics, 61*(2), 101–118. doi:10.1002/nav.21569.

Bortfeld, A., & Forster, F. (2012). A tree search procedure for the container premarshalling problem. *European Journal of Operational Research, 217*, 531–540. doi:10.1016/j.ejor.2011.10.005.

Carlo, H. J., Vis, I. F. A., & Roodbergen, K. J. (2014). Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research, 235*, 412–430. doi:10.1016/j.ejor.2013.10.054.

Caserta, M., Schwarze, S., & Voß, S. (2011a). Container rehandling at maritime container terminals. In J. W. Böse (Ed.), *Handbook of terminal planning*. In *Operations Research/Computer Science Interfaces Series: vol. 49* (pp. 247–269). Springer.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research, 219*, 96–104. doi:10.1016/j.ejor.2011.12.039.

Caserta, M., Voß, S., & Sniedovich, M. (2011b). Applying the corridor method to a blocks relocation problem. *OR Spectrum, 33*(4), 915–929. doi:10.1007/s00291-009.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, M. (2012). Premarshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications, 39*, 8337–8349. doi:10.1016/j.eswa.2012.01.187.

Galle, V., Manshadi, V. H., Borjian Boroujeni, S., Barnhart, C., & Jaillet, P. (2018). The stochastic container relocation problem. *Transportation Science, 52*(5), 1035–1058. doi:10.1287/trsc.2018.0828.

Hussein, M., & Petering, M. E. H. (2012). Genetic algorithm-based simulation optimization of stacking algorithms for yard cranes to reduce fuel consumption at seaport container transshipment terminals. In *Proceedings of the 2012 IEEE congress on evolutionary computation* (pp. 1–8). doi:10.1109/CEC.2012.6256471.

Ji, M., Guo, W., Zhu, H., & Yang, Y. (2015). Optimization of loading sequence and rehandling strategy for multi-quay crane operations in container terminals. *Transportation Research Part E, 80*, 1–19. doi:10.1016/j.tre.2015.05.004.

Jovanovic, R., Tanaka, S., Nishi, T., & Voß, S. (2019a). A GRASP approach for solving the blocks relocation problem with stowage plan. *Flexible Services and Manufacturing Journal, 31*, 702–729. doi:10.1007/s10696-018-9320-3.

Jovanovic, R., Tuba, M., & Voß, S. (2017). A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research, 25*(1), 1–28. doi:10.1007/s10100-015-0410-y.

Jovanovic, R., Tuba, M., & Voß, S. (2019b). An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research, 274*, 78–90. doi:10.1016/j.ejor.2018.09.038.

Kim, K. H., & Hong, G. (2006). A heuristic rule for relocating blocks. *Computers & Operations Research, 33*(4), 940–954. doi:10.1016/j.cor.2004.08.005.

Ku, D., & Arthanhari, T. S. (2016). Container relocation problem with time windows for container departure. *European Journal of Operational Research, 252*, 1031–1039. doi:10.1016/j.ejor.2016.01.055.

Lee, C.-Y., & Song, D. P. (2017). Ocean container transport in global supply chains: Overview and research opportunities. *Transportation Research Part B, 95*, 442–474. doi:10.1016/j.trb.2016.05.001.

Lee, Y., & Hsu, N. Y. (2007). An optimization model for the container pre-marshalling problem. *Computers & Operations Research, 34*(11), 3295–3313. doi:10.1016/j.cor.2005.12.006.

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research, 239*, 297–312. doi:10.1016/j.ejor.2014.03.011.

Lin, D.-Y., Lee, Y.-L., & Lee, Y. (2015). The container retrieval problem with respect to relocation. *Transportation Research Part C, 52*, 132–143. doi:10.1016/j.trc.2015.01.024.

Parreño-Torres, C., Alvarez-Valdes, R., & Ruiz, R. (2019). Integer programming models for the pre-marshalling problem. *European Journal of Operational Research, 274*, 142–154. doi:10.1016/j.ejor.2018.09.048.

Rendl, A., & Prandtstetter, M. (2013). Constraint models for the container pre-marshalling problem. In G. Katsirelos, & C. G. Quimper (Eds.), *Proceedings of the 12th international workshop on constraint modelling and reformulation, ModRef 2013:* (pp. 44–56).

Scholl, J., Boywitz, D., & Boysen, N. (2018). On the quality of simple measures predicting block relocations in container yards. *International Journal of Production Research, 56*(1–2), 60–71. doi:10.1080/00207543.2017.1394595.

da Silva Firmino, A., de Abreu Silva, R. M., & Times, V. C. (2019). A reactive GRASP metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics, 25*(2), 141–173. doi:10.1007/s10732-018-9390-0.

Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: A literature update. *OR Spectrum, 30*(1), 1–52. doi:10.1007/s00291-007-0100-9.

Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. *OR Spectrum, 26*(1), 3–49. doi:10.1007/s00291-003-0157-z.

Tanaka, S., & Mizuno, F. (2018). An exact algorithm for the unrestricted block relocation problem. *Computers & Operations Research, 95*, 12–31. doi:10.1016/j.cor.2018.02.019.

Tanaka, S., & Tierney, K. (2018). Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. *European Journal of Operational Research, 264*, 165–180. doi:10.1016/j.ejor.2017.05.046.

Tierney, K., Pacino, D., & Voß, S. (2017). Solving the pre-marshalling problem to optimality with A* and IDA*. *Flexible Service and Manufacturing Journal, 29*, 223–259. doi:10.1007/s10696-016-9246-6.

Tierney, K., & Voß, S. (2016). Solving the robust container pre-marshalling problem. In A. Paias, M. Ruthmair, & S. Voß (Eds.), *Computational logistics*. In *Lecture Notes in Computer Science 9855* (pp. 131–145).

Voß, S., & Schwarze, S. (2019). A note on alternative objectives for the blocks relocation problem. In C. Paternina-Arboleda, & S. Voß (Eds.), *Proceedings of the Computational logistics. ICCL 2019*. In *Lecture Notes in Computer Science: vol. 11756*. Springer.

Wu, K. C., & Ting, C. J. (2012). A beam search algorithm for minimizing reshuffle operations at container yards. In *Proceedings of the 2010 international conference on logistics and maritime systems*.

Zehender, E., Caserta, M., Feillet, D., Schwarze, S., & Voß, S. (2015). An improved mathematical formulation for the blocks relocation problem. *European Journal of Operational Research, 245*, 415–422. doi:10.1016/j.ejor.2015.03.032.

Zhao, W., & Goodchild, A. V. (2010). The impact of truck arrival information on container terminal rehandling. *Transportation Research Part E, 46*, 327–343. doi:10.1016/j.tre.2009.11.007.

Please cite this article as: B.G. Zweers, S. Bhulai and R.D. van der Mei, Optimizing pre-processing and relocation moves in the Stochastic Container Relocation Problem, European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2019.11.067