Pre-processing a container yard under limited available time

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

[a] *Centrum Wiskunde & Informatica, Stochastics, the Netherlands*
[b] *Vrije Universiteit Amsterdam, Department of Mathematics, the Netherlands*

Abstract

To pick up a container from a container terminal, other containers may need to be relocated to other positions. In practice, these relocation moves are usually done when it is busy at a terminal. However, if the crane is idle for some amount of time, it may be more efficient to execute some pre-processing moves to reduce the number of future relocation moves. In this paper, we propose a model for optimal pre-processing moves if the available time is limited. We develop a heuristic to produce fast solutions for this new optimization problem. This heuristic consists out of multiple phases and for each phase, two different approaches are possible, and thus, the heuristic produces multiple solutions. Besides that, an optimal branch-and-bound algorithm is presented. Third, we propose another heuristic in which the remaining relocation moves are estimated in a sub-optimal way in the branch-and-bound method. This algorithm is not guaranteed to find the optimal solution, but its running time is faster than the optimal branch-and-bound method. Finally, we give an integer linear program that can be used to extend these solutions for a single bay of containers to a complete yard of containers.

Keywords: Terminal operations; Container Relocation Problem; Container Pre-Marshalling Problem; Branch-and-bound; Heuristic

1. Introduction

Nowadays, it is hard to imagine a world without containers. About 75% of the total global volume of goods is shipped by sea, and almost half of this sea trade is shipped in shipping containers (Lee & Song, 2017). Shipping goods in a container is so popular because it is much easier to transship goods from one mode of transportation to the other when they are in a container. Although different goods generally have different sizes and weights, containers have standardized dimensions. Therefore, the container terminals at which the containers are transshipped have specialized equipment to handle these containers efficiently. A result of this easy transshipment is that for each part of the trip the most efficient mode of transportation for a container can be used. For instance, deep-sea vessels are used to transport thousands of containers from one continent to the other. Shipping that many containers is only possible if there is an efficient way to ship the containers from the deep-sea port to their final destination.

The part of the transportation from the deep-sea port to its final destination is called the *inland transportation* of a container. Usually, there are three possible modes for inland transportation: trucks,

*Corresponding author
Email address: b.g.zweers@cwi.nl (Bernard Zweers)

trains, and barges. Trucks are often used if the container's destination is close to the container terminal. However, if the final destination is further away, it might be cheaper to transport the containers with barges or trains to an inland terminal. These barges or trains can ship up to a few hundred containers to an inland container terminal. These inland terminals are often located in the vicinity of the final destination, and thus a truck can be used to ship a specific container to its destination. Besides the fact that barge and train transportation is cheaper, it also has a lower CO_2-emission per container than truck transportation. However, using barges and trains also causes an extra transshipment at an inland container terminal.

The transshipment of goods in a container still imposes extra costs, even though it is cheaper than not using containers at all. First of all, there are costs concerning the equipment that is used to move a container from one mode of transportation to the other. Furthermore, the arrival and departure times of different transportation modes are, in general, not synchronized and containers need to be stored temporarily at the container terminal. To reduce the associated storage costs, space is saved by stacking containers on top of each other. We refer to a single row of containers as a *bay* and a set of bays is called a *yard*. The equipment that is used to handle the containers in a terminal can only access the top container of a stack. Hence, ideally, a container that needs to leave the terminal is always placed on top of a stack at its departure time. However, when a container is placed in a stack, it is usually unknown when it has to leave the yard. Therefore, it often happens in practice that another container is on top of the container that needs to be retrieved. This so-called *blocking container* needs to be relocated to another stack before the *target container* can be retrieved. This move is called a *relocation move* and imposes extra costs. For instance, the waiting costs that are encountered by the truck that will transport that container. Moreover, since the total number the containers that are shipped is increasing (United Nations Conference on Trade And Development, 2020), the operations at a container terminal are under pressure to have an efficient transshipment of containers to avoid a delay. Hence, these unproductive relocation moves should be avoided as much as possible.

Although the departure time of a container is usually unknown when a container arrives at a terminal, a few hours before containers are retrieved, it is known in which period they will be retrieved. If at that time the crane is idle, then *pre-processing moves* can be performed. The idea of pre-processing moves is to move containers to reduce the number of future relocation moves. As these pre-processing moves can be done when the crane is otherwise idle, the costs are lower, because no truck is waiting. However, in practice, the crane is only idle for a limited time, thus the number of pre-processing moves that can be performed is limited. This study is motivated from real-life practice: an inland terminal at the port of Amsterdam is facing the problem of determining which pre-processing moves to perform to obtain the biggest decrease in the number of relocation moves. To solve this problem, we present in this paper a new optimization problem: the *Stochastic Container Relocation Problem with Constrained Pre-Processing* (SCRPCPP).

The contribution of this work is fourfold. First, we introduce a new optimization problem faced by a real inland container terminal. Second, we develop two heuristics to solve this problem for a single bay of containers. Third, we present an optimal algorithm for the same problem. Finally, a method to optimally extend the solution for single bays to multiple bays is presented.

The remainder of this paper is organized as follows. We start with an overview of related work in the literature in Section 2. Afterward, the SCRPCPP is formally defined in Section 3. Subsequently, in Section 4 two heuristic methods and an optimal algorithm to solve the SCRPCPP are discussed. In Section 5, we extend the SCRPCPP from a single bay to multiple bays and present a solution method. We compare all solution methods using numerical experiments in Section 6. Finally, we conclude this paper in Section 7.

2. Literature review

Many different problems concerning the stacking of containers have been considered in the literature. An overview of all these different problems can be found in Carlo et al. (2014), Caserta et al. (2011), and Lehnfeld & Knust (2014). Below, we discuss the literature concerning the three problems that are closest to our problem, namely the Stochastic Container Relocation Problem (SCRP), the Container Pre-Marshalling Problem (CPMP), and the Unrestricted Blocks Relocation Problem (UBRP). All these problems consider only containers in a single bay, only a few papers have been devoted to problems within multiple bays.

In the SCRP, each container has a time interval in which it is retrieved from the yard. If it has containers on top of it when it is retrieved, then these blocking containers need to be relocated to other stacks. The goal of the SCRP is to relocate the containers to a stack in which their expected number of remaining relocation moves is minimized. We know that each container is retrieved inside its interval, but multiple containers could be retrieved in the same interval and the retrieval order inside an interval is unknown. The SCRP was introduced by Zhao & Goodchild (2010). They study the 'value of information' for this problem and propose a simple heuristic to solve the SCRP. In Ku & Arthanhari (2016), a more advanced heuristic is proposed which is called the Expected Reshuffle Index (ERI) heuristic. In this heuristic, the expected number of reshuffles needed to retrieve the containers from each stack is calculated. The container is relocated to the stack for which this number is the lowest. Another heuristic for the SCRP, the so-called Expected Minmax (EM) heuristic, is proposed by Galle et al. (2018b). This heuristic first tries to place a container in a stack in which it does not need to be relocated. If such a stack does not exist, a container is placed in a stack in which its relocation move will be the latest. In Galle et al. (2018b), also an optimal formulation for the SCRP is given which can solve small instances within a reasonable time. The solution of the EM heuristic is close to the optimal solution and outperforms the ERI heuristic of Ku & Arthanhari (2016). If each interval only contains a single container, the problem is referred to as the Container Relocation Problem (CRP), because there is no more stochasticity. The CRP is sometimes also referred to as the Blocks Relocation Problem (BRP), because it can also be applied to many other last-in-first-out systems in which one can only retrieve a single item from a stack or row. The deterministic variants, CRP and BRP, have been studied extensively, see for example Caserta et al. (2012), Zehender et al. (2015) and Jovanovic et al. (2019).

The difference between the SCRP and the UBRP is that in the latter not only a blocking container may be relocated, but each container is allowed to be relocated. The first mathematical formulation for the UBRP is given in Caserta et al. (2012). In Petering & Hussein (2013), an Integer Linear

Program (ILP) formulation that uses fewer decision variables than the model in Caserta et al. (2012) is proposed. Moreover, the running time of the model of Petering & Hussein (2013) is faster. At the moment, the best exact algorithm for the UBRP can be found in Tanaka & Mizuno (2018), in which an exact branch-and-bound algorithm for the UBRP is presented. Besides the ILP formulation, in Petering & Hussein (2013), also a look-ahead heuristic is proposed. In Jin et al. (2015), a heuristic that constructs a partial tree with possible layouts is constructed. The layout for which a greedy heuristic gives the lowest number of relocation moves is selected as the best solution. In Tricoire et al. (2018), a metaheuristic for the UBRP is presented in which a beam search is combined with constructive heuristics. Finally, Feillet et al. (2019) present a local search heuristic for the UBRP that is incorporated in a dynamic programming formulation.

Contrary to the SCRP and the UBRP, in the CPMP, the containers are moved before any container is retrieved and these moves are called *pre-marshalling moves*. The goal of the CPMP is to use as few pre-marshalling moves as possible to obtain a stacking of the containers in which no relocation moves are needed. One way to solve the CPMP is to use ILP models. In the first paper about the CPMP, Lee & Hsu (2007) model the CPMP as a multi-commodity flow network that they solve using an ILP formulation. In de Melo da Silva et al. (2018), an ILP formulation is presented that can solve both the CPMP as the CRP. In Parreño-Torres et al. (2019), eight different ILP formulations for the CPMP are given, which are currently the best mathematical models for the CPMP. Tree-based methods are another way to solve the CPMP to optimality. The first tree-based method was an A^* algorithm introduced by Expósito-Izquierdo et al. (2012). An A^* algorithm is a variant of a branch-and-bound algorithm in which also the expected costs of the nodes below the current node are estimated. In Tierney et al. (2017), an improved A^* algorithm is given in which they make use of the lower bounds for the CPMP of Bortfeld & Forster (2012). The current state-of-the-art algorithm for solving the CPMP to optimality is a branch-and-bound algorithm that is first presented by Tanaka & Tierney (2018) and later improved by Tanaka et al. (2019). This method can solve almost all real-sized instances within an hour. Although the CPMP has never been proven to be NP-hard, no faster optimal algorithms exist for the CPMP. Hence, besides optimal algorithms, also heuristics have been proposed for this problem.

A constructive heuristic is the Lowest Priority First heuristic of Expósito-Izquierdo et al. (2012). The idea of this heuristic is to place the containers with the largest time frames in the correct position first. This heuristic consists of different phases and in Jovanovic et al. (2017), multiple heuristics are applied to every single of these phases. Using this approach, multiple different solutions are obtained and the best is chosen, which results in better solutions than the heuristic of Expósito-Izquierdo et al. (2012). In Hottung & Tierney (2016), a genetic algorithm is used to find the best parameters of a constructive heuristic. In Hottung et al. (2020), a tree search heuristic for the CPMP is developed in which the branching decisions are made based on a model learned by a deep neural network. The results of this heuristic are better than the heuristic of Hottung & Tierney (2016), but at the expense of longer computation times.

In the papers described above, the objective has been to minimize the number of pre-marshalling or relocation moves. Nevertheless, one can also decide to minimize the time needed to perform these moves. In Voß & Schwarze (2019), it has been shown that if this objective is chosen, then the number

of relocation moves is often also minimal. In case the time needed is in the objective function, it is also natural to allow movement of containers to multiple bays. The first paper that includes a time dimension in the objective function is Lee & Lee (2010). In this paper, the objective is a weighted sum of the number of relocation moves and the time needed to perform these relocation moves. Lee & Lee (2010) propose a three-phase heuristic to solve this problem, but even for small instances the running time is too large to be used in practice. A much faster heuristic that also has a higher solution quality is proposed in Lin et al. (2015). This heuristic calculates for each stack a weighted sum of the minimum time frame of that stack and the time needed to travel to that stack. The stack for which this index is the lowest is chosen as destination stack for the container. In da Silva Firmino et al. (2019), an optimal A^* algorithm and a GRASP heuristic are proposed to minimize the crane's working time. A GRASP heuristic is a meta-heuristic in which a greedy construction heuristic is combined with a local search heuristic. In Galle et al. (2018a), a model is considered in which only the retrieval order of the first set of containers is known. The objective is to minimize a weighted sum of the time needed to retrieve these containers and the future remaining number of relocation moves in the bay. In Casey & Kozan (2012), a model is studied in which the objective is to minimize the total time needed to perform all movements. They consider both incoming and outgoing containers and propose both constructive and meta-heuristics to solve the problem. However, they study a terminal that uses a straddle carrier to handle containers, which is a different type of equipment than used in the other papers mentioned.

The concept of pre-processing a container bay to reduce the number of relocation moves was first introduced by Zweers et al. (2020). In this paper, the weighted sum of pre-processing moves and expected relocation moves is minimized and with that it generalizes both the SCRP and the CPMP: if the weight of a pre-processing move is set close to zero, then the problem is equivalent to the CPMP and the problem is equivalent to the SCRP if the weight of a pre-processing move is set extremely high. The pre-processing phase could also be seen as a variant of the UBRP in which all containers can be moved, but only before the very first container is retrieved.

3. Problem formulation

To formulate the SCRPCPP, it is important to describe the processes at a container terminal first. In container terminals, containers are stored in a rectangular yard, as illustrated in Figure 1(a). One row of containers in such a yard is called a bay. The commonly used equipment to store and retrieve a container from a yard is a Rail Mounted Gantry Crane (RMGC), which is also given in Figure 1(a). Other vehicles, for instance a terminal tractor or a truck, are used to move the container to another position inside or outside the terminal, such as a ship or a customer. In both figures of Figure 1, such a vehicle is depicted. In Figure 1(b), one sees one bay and an RMGC with a trolley attached to it. The trolley can be lowered to pick up a container from the yard, but only the top container of a stack can be picked up by the trolley. Afterward, the trolley is lifted again to move the container over the other containers to the end of the bay to place the container at the vehicle. As a result, the maximum number of containers in a stack is limited, because otherwise, the trolley cannot move a container from one side of the bay to the other. Moreover, the width of the RMGC also imposes a constraint

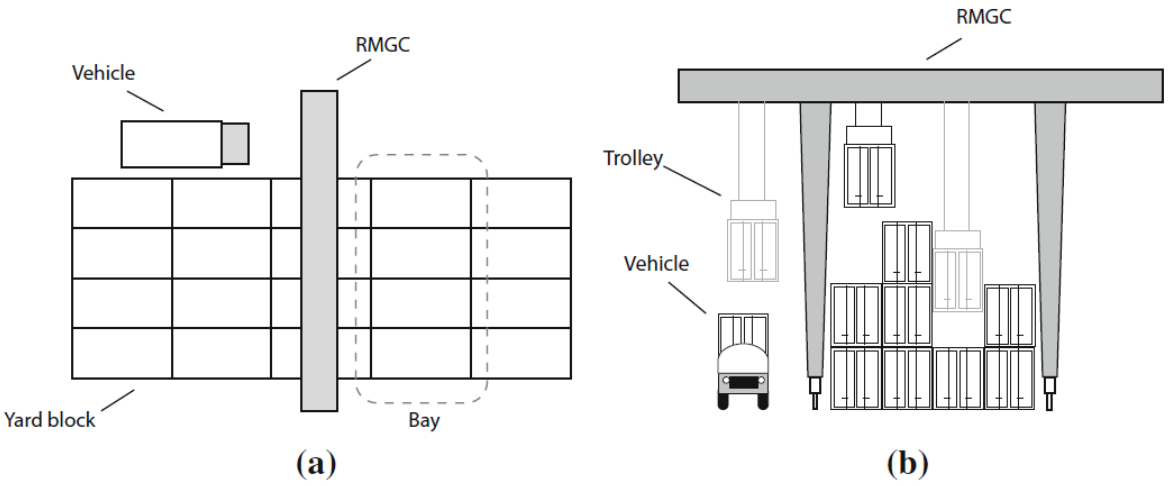


Figure 1: The layout of an RMGC and a container yard from above in (a) and from the front in (b) (Tierney et al., 2017).

on the maximum number of stacks in a bay. Consequently, the number of containers in a bay, called *slots*, is also limited. For example, the bay in Figure 1(b) can contain at most four stacks that have a maximum height of three and therefore has twelve slots.

The RMGC is also used to move the top container of a stack to another stack if it has to be relocated. In case a container is moved to a stack in another bay, the entire RMGC has to move, whereas only the trolley has to move if the container is moved to a stack in the same bay. The former is much more time-consuming, and on top of that, in some terminals, it is not even allowed to move the crane if it is carrying a container (Lee & Hsu, 2007). Therefore, in the SCRPCPP it is not allowed to move a container from one bay to another bay and thus, the problem is two-dimensional, similar to Figure 1(b). As will be explained later, the SCRPCPP can be extended to multiple bays, but also in that scenario no containers are moved from one bay to another.

The SCRPCPP is inspired by a problem faced by an inland terminal in the port of Amsterdam. For all containers that will leave that terminal on a certain day, it is known at the beginning of that day in which time interval they are retrieved. However, the exact arrival time of the truck on which the container is transported is unknown, because its departure at its previous visit and its travel time are stochastic. In practice, the time interval is often about one hour and multiple trucks can arrive during this interval. Hence, we do not know the exact order of the arrival of the trucks in that period.

In general, a crane at an inland terminal is less busy than a crane at a larger deep-sea terminal. If the crane is idle, it can perform some pre-processing moves to reduce the number of relocation moves. However, the time needed to reshuffle the containers such that no relocation moves are needed is often more than the idle time of the crane. Therefore, it is impossible to perform *all* pre-marshalling moves and it is important to perform the pre-processing moves that reduce the number of relocation moves the most. The SCRPCPP consists out of two different phases: (i) the *pre-processing* and (ii) the *relocation phase*. The pre-processing phase ends the moment when the first container is retrieved and at that time the relocation phase starts. The relocation phase is equivalent to the SCRP. At first sight, the moves in the pre-processing phase might look the same as the pre-marshalling moves

in the CPMP. However, the main difference is that after all pre-marshalling moves are performed no relocation moves are needed and there might still be some relocation moves after the pre-processing phase has finished.

The remainder of this section is organized as follows. In Section 3.1, the assumptions that are listed for the SCRPCPP are explained. Using these assumptions, the mathematical formulation of the SCRPCPP is given in Section 3.2. Finally, we extend the method for a single bay to multiple bays in Section 5.1.

3.1. Assumptions

We have decided to make a few assumptions to keep the SCRPCPP tractable while respecting the current practice at container terminals as much as possible. The assumptions are the same as the ones made in Zweers et al. (2020). In this section, we explain why these assumptions are realistic and what kind of problems arise if they are relaxed. The assumptions are as follows:

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 any other stack does not depend on the stack to which a container is moved.
5. Each container can be placed on top of every other container.

Assumption 1 is made because it imposes a natural end of the period that needs to be considered. Moreover, most container terminals reserve empty stacks for containers unloaded from one specific vessel. On top of that, they have specific areas in which only outbound or inbound containers are stored. As a consequence, it often happens that no new containers arrive in a bay with outbound containers that will leave the terminal this day. A problem that is closely related to our problem and in which new incoming containers in a bay are allowed is the *Dynamic Container Relocation Problem* (Akyüz & Lee, 2014).

In Section 2, we have already seen that Assumption 2 is sometimes relaxed and that then the unrestricted version of the CRP is obtained. However, this second assumption makes the relocation phase significantly easier and it is also common practice in container terminals (Caserta et al., 2012). Under Assumption 2, the number of relocation moves that are performed when a single container is retrieved is the same as the number of containers that are blocking this target container. Whereas in the unrestricted version, there is no tight bound for the number of relocation moves. On top of that, with this assumption it is straightforward to check if a container does not need to be moved in the relocation phase: if a container has only containers underneath it that are picked up later, then it will never be moved in the relocation phase. We say that a container for which no relocation moves are needed is *well-placed* or *correctly-placed*. Contrary, a container that is not well-placed is called *poorly-placed*.

The third assumption distinguishes the CRP from the SCRP, because in the CRP each container has its unique time interval in which it is picked up. This assumption reflects the situation in which terminals have a truck appointment system (Ku & Arthanhari, 2016). In such a system the terminal has a fixed number of time intervals in which trucks can arrive to pick up a container. Multiple trucks can make an appointment to pick up a container in such a time interval. After that, the terminal knows which containers will leave the terminal in which interval, but it does not know anything about the order of the departures in such a time interval. As no information is available, it is most natural to assume that the retrieval order is a uniform permutation, since then each order is equally likely. The moment the information of the retrieval order inside an interval becomes known to the terminal can vary and leads to two slightly different variants of the SCRP: the *online model* (Zhao & Goodchild, 2010; Ku & Arthanhari, 2016) and the *batch model* (Galle et al., 2018b). In the batch model, a container is only retrieved after all trucks for that time interval have arrived at the terminal. Consequently, if the first container in a time interval is retrieved, then the exact retrieval order for all containers in that time interval is known. Whereas in the online model, each container is retrieved immediately after a truck has arrived to pick it up. Therefore, no extra information about the containers that are retrieved later during that interval is known when this container needs to be relocated. The batch model is more suitable for large terminals in which the time intervals are short and in which there is a significant amount of time between the arrival of a truck at the terminal and the moment it is served, whereas the online model better reflects smaller terminals in which trucks are served faster (Galle et al., 2018b). As the batch model implicitly assumes that the crane does not have any idle time, the online model is a variant in which pre-processing makes more sense. Motivated by this, in this paper we use the online model.

Assumption 3 is the only assumption that makes the SCRPCPP stochastic. The stochasticity lies in the fact that it is not completely known whether a container needs to be relocated or not. Furthermore, if it needs to be relocated, then it is unknown how often it will be relocated. As a result of Assumption 2, a container may only be relocated if it is blocking the container that is picked up. However, if that target container is the first container of the interval, the bay might look completely different at that retrieval moment than if it were the last container being retrieved in that interval. Consequently, Assumption 3 implies that computing the expected number of relocation moves for a given bay and relocation policy is computationally expensive. On the other hand, if a certain relocation policy for the deterministic CRP is used, then the number of relocation moves for a specific bay can be computed efficiently.

The trolley needs to be positioned exactly above the container to pick it up which is a very precise task. Moreover, placing a container on top of another container is also hard. As a result, picking up and putting down a container is much more time-consuming than actually moving the trolley to another stack, and thus Assumption 4 is realistic. A consequence of Assumption 4 is that the only objective to minimize is the number of relocation moves. Since the costs of relocating a container do not depend on the distance between the stack, one could see two bays in which the stacks of one bay are a permutation of the stacks of another bay as equivalent. In Section 2, we have seen that there are a few papers in which Assumption 4 is not made and the total objective is to minimize the weighted average of the number of relocation moves and the total working time of the crane.

B	Specific layout of a bay
$\mathcal{S}$	Set of all stacks in a bay
$B(s_1, s_2)$	Bay after moving the top container of stack s_1 in bay B to stack s_2
$C(t, s)$	Container that is positioned in stack s at tier t
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
$s(c)$	Stack of container c
$u(c)$	Smallest time frame of containers underneath c
$t(c)$	Time frame of container c
$p(c)$	Position of a container c in stack $s(c)$: 1 is the lowest and H the highest position
p_j	The j^{th} pre-processing moves
$\mathcal{P}$	Set of all pre-processing moves
ρ	Maximum number of pre-processing moves
π	Relocation policy
LB	Lower bound for the optimal solution
$f(B, \pi)$	Estimation of expected number of relocation moves using policy π in bay B
τ	Time needed to move a container within a bay from one stack to the other
T	Total time available for the pre-processing phase
b	Number of bays in a yard

Table 1: Notation for the SCRPCPP.

Although containers have standardized sizes, the number of different sizes for containers is still quite large. Containers of different sizes can, in general, not be stacked on top of each other. Nevertheless, in most terminals, containers with different sizes are not placed in the same bays and thus Assumption 5 is still realistic. Moreover, if there would be a container in a bay on which a container could not be placed, then the possible number of stacks with a feasible relocation move is lower and the SCRPCPP is only easier.

3.2. Mathematical formulation

In this section, we use the five assumptions and the notation of Table 1 to define the SCRPCPP formally. A pre-processing move p_j is the j^{th} pre-processing move that is performed and it is uniquely defined by its *origin stack* o_j and its *destination stack* d_j, i.e., $p_j = (o_j, d_j)$. Let $\mathcal{P} := (p_1, p_2, \ldots, p_m)$ be an ordered set with the set of performed pre-processing moves, such that for each $1 \leq i < j \leq m$ move p_i is performed before move p_j. If the pre-processing moves in $\mathcal{P}$ are applied to a bay B, then the resulting bay is unique and given by $B(\mathcal{P})$. As a consequence of Assumption 3, we assume that there is a fixed amount of time needed to do a single pre-processing move. Let us denote this time by τ. Moreover, we assume that the available time for the pre-processing phase is known and given by T. Using these two time units, there is a maximum number of pre-processing moves that can be performed, namely $\rho := \left\lfloor \frac{T}{\tau} \right\rfloor$. Note that if ρ is defined in this way, it is an integer.

Let n be the number of different retrieval orders in a bay and let $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ be the set of those orders. We assume that the actions in the relocation phase are made according to a certain relocation policy π that potentially takes the complete bay and all possible retrieval orders into consideration. Given a relocation policy π and a retrieval order σ_i, the number of relocation moves for a bay B is denoted by $R_\pi(B, \sigma_i)$. Hence, the expected number of relocation moves for a bay B is given by $\frac{1}{n} \sum_{i=1}^{n} R_\pi(B, \sigma_i)$. Using this notation, the SCRPCPP is given by the following

mathematical optimization problem:

$$\min_{\pi,P} \quad \frac{1}{n}\sum_{i=1}^{n} R_\pi(B(\mathcal{P}),\sigma_i) \tag{1}$$

$$\text{subject to:} \quad |\mathcal{P}| \leq \rho. \tag{2}$$

The objective function in (1) represents the expected number of relocation moves for bay B after the pre-processing moves in $\mathcal{P}$ if policy π is applied to the relocation phase. Furthermore, the constraint in (2) ensures that no more than ρ pre-processing moves are performed. In case ρ is less than one, the formulation above is exactly equivalent to the SCRP, because then no pre-processing move can be performed. In Caserta et al. (2012), it has been shown that the deterministic CRP is NP-hard. Since the SCRP is a generalization of the CRP and the SCRPCPP is equivalent to the SCRP if no pre-processing moves are allowed, the SCRPCPP is also NP-hard.

Running example

Throughout this paper, the bay given in Figure 2 will be used to illustrate concepts and solution methods. The bay of Figure 2 consists out of five stacks and we assume that in this bay containers can be stacked at most four containers high. In the center of the thirteen squares indicating containers is an integer from 1 up to 5 shown. This number indicates the time interval in which a container is retrieved: the containers with a 1 inside them are the first to be retrieved and in the last interval containers that are indicated by a 5 are picked up. Although it is easy to check that a container is relocated at least once, computing the exact number of expected relocation moves for a container is difficult. As then, one also needs to be able to compute the bay at the moment that the container has to be relocated and that, in turn, depends on the relocation moves did during the first time interval. For example, the container with time frame 5 in stack 4 is relocated at least once, because the container underneath it is retrieved in time interval 2. However, it is hard to consider all potential bays at the moment that this container is relocated. On the other hand, computing the exact expected number of relocation moves for the top container of the first stack is easy. The retrieval order of the containers in a time interval is a random uniform permutation, and thus, both the probability that the top container of stack 1 is retrieved before and after the bottom container is $\frac{1}{2}$. One can easily check that there should be an available slot in a stack not containing a container with time frame 4 if the top container has to be relocated. Hence, it will at most be relocated once and its expected number of relocation moves is $\frac{1}{2}$.

If only a single pre-processing move is allowed for this bay, the best pre-processing move is moving the top container of the fourth stack to the first stack. The top container of stack 4 is poorly-placed in that stack and will need to be relocated at least once. However, all containers in the first stack have a higher time frame than 3, so the container will not need to be relocated if it is placed in stack 1. The top container of stack 3 can also be placed such that it will not be relocated in stacks 1, 4 and 5, but its expected number of relocation moves is only $\frac{1}{2}$ in the example bay of Figure 2, so less improvement is made by moving this container. Nevertheless, if two pre-processing moves are allowed, this container will be moved. If more than two pre-processing moves are allowed for the bay of Figure

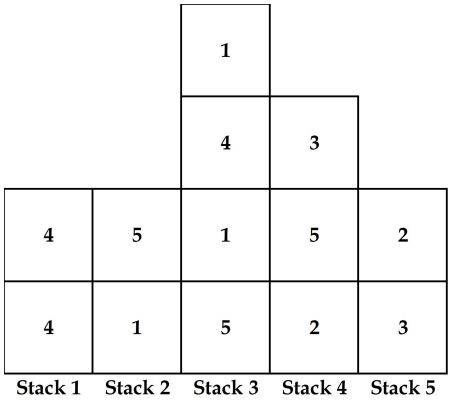


Figure 2: Layout of a bay with five stacks that have a maximum height of four containers (Zweers et al., 2020).

2, it is more complicated to determine the best pre-processing moves. We will discuss that situation later.

4. Solution methods

In this section, we discuss three solution methods to solve the SCRPCPP. We only develop methods for the pre-processing phase, because a good heuristic and optimal formulation for the relocation phase have already been given in Galle et al. (2018b). As we have shown in Section 3.2, the SCRPCPP is NP-hard, thus we first give a heuristic for the pre-processing phase in Section 4.1. In this heuristic, we need a method to estimate the number of relocation moves for a given bay. We use the *rule-based method* presented in Zweers et al. (2020) to obtain the estimation. The idea behind this estimation method is to use three rules to estimate the expected number of relocation moves for a single container. Using these rules, we can divide the containers into four different groups, namely containers for which the estimated number of expected relocation moves is: (i) zero, (ii) strictly between zero and one, (iii) exactly one, and (iv) more than one. Every container in such a group gets a weight of, respectively, 0, 0.5, 1, or 1.4 and the weighted sum of all containers is the estimate for the total number of expected relocation moves. After we have developed a heuristic, we derive in Section 4.2 a lower bound for the SCRPCPP. This lower bound is used in a branch-and-bound method that is presented in Section 4.3. In this branch-and-bound algorithm, the number of expected relocation moves for a bay needs to be calculated. If the optimal expected number of relocation moves is calculated in each node of the branch-and-bound tree, then the branch-and-bound algorithm produces the optimal solution for the SCRPCPP. However, the number of relocation moves for a bay can also be estimated in a sub-optimal way using the rule-based method and that gives us another heuristic for the SCRPCPP.

4.1. Top Correct heuristic

In this section, a heuristic to find good pre-processing moves is described. We call this heuristic the *Top Correct* (TC) *heuristic*. The general idea of the TC heuristic is that we try for each combination of two stacks, to move the top container of one stack, called the *origin* stack, to the correct position in another stack, called the *destination* stack. It is also possible to place a container in a correct position in the same stack as it is currently located. In this case, the origin and destination stack are the same

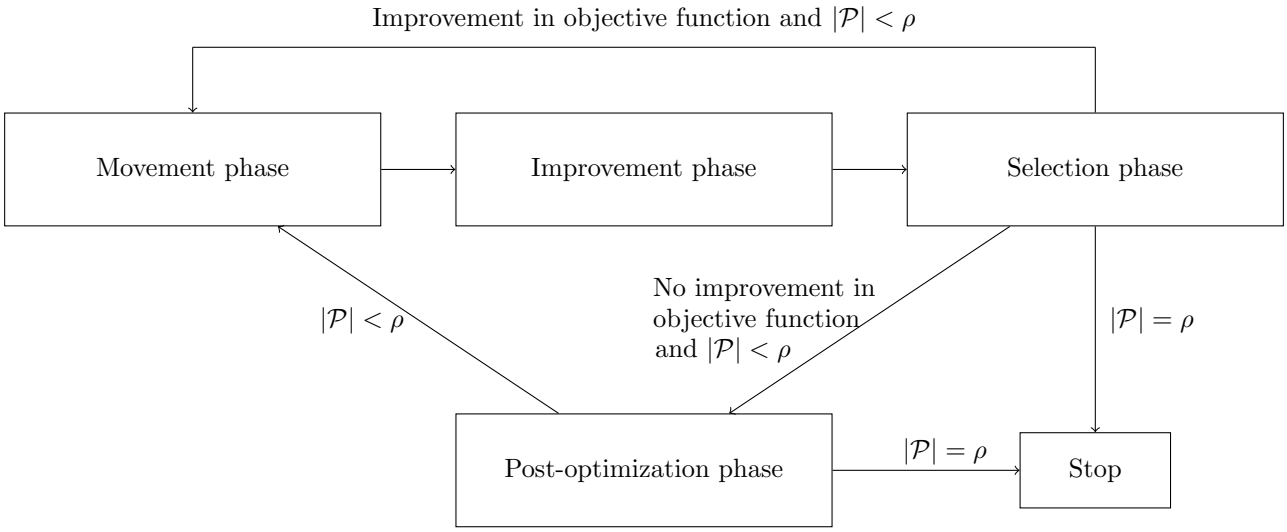


Figure 3: Illustration of the TC heuristic.

stack. After that, we calculate for each combination of origin and destination stacks the difference in the number of expected relocation moves and choose the pair that yields the largest decrease in the expected number of relocation moves without exceeding the maximum number of pre-processing moves. In total, the TC heuristic consists of four different phases that we call (i) the *movement*, (ii) the *improvement*, (iii) the *selection*, and (iv) the *post-optimization* phase. In the first three stages, two different decisions can be made, thus the heuristic has $2^3 = 8$ different variants. Below, the four different phases are described. In Figure 3, an illustration of the TC heuristic is given.

Phase 1: Movement

In the movement phase, the top container of the origin stack is moved into a correct position in the destination stack. To place the container correctly in the destination stack, it might be necessary to move containers from the destination stack to other stacks before the top container of the origin stack can be moved to the destination stack. We refer to these moves as the *cleaning moves*. In cleaning moves, containers are ideally moved to stacks in which they are correctly-placed. Out of the stacks in which a container is correctly-placed, the stacks with the fewest poorly-placed containers are selected. We prefer stacks in which few containers are poorly-placed, because if a container is placed in a stack in which it is correctly-placed, while other containers are not correctly-placed, then it might still need to be moved a second time in the pre-processing phase. If there are multiple stacks with the same number of poorly-placed containers and in which the container that is moved will be correctly-placed, then the container is moved to stack in which the minimum time frame is the lowest. If it is not possible to move a container in the cleaning phase to a stack in which it will not need to be relocated a second time, then it is moved to a stack in which it is moved as late as possible in the relocation phase. This last step is equivalent to the procedure applied in the EM heuristic for the SCRP (Galle et al., 2018b).

We have defined that a container is correctly-placed if it only has containers with a lower time frame underneath it. However, if for a specific container the minimum time frame of all containers

underneath it is the *same* as its own time frame, then the probability that it does not need to be relocated is positive but smaller than one. We call this type of container *semi-correct*. In case the top container of the origin stack is allowed to be semi-correctly-placed in the destination stack, it could be that fewer cleaning moves are necessary, while still the number of relocation moves is reduced. Hence, the two different variants for the movement phase are whether we require the top container of the origin stack to be semi-correct or correct in the destination stack.

Phase 2: Improvement

It could be that after the movement phase also other containers can be correctly-placed. For instance, when the container that is correctly-placed in the movement phase has a large time frame, many containers can be stacked upon that container. So after the movement phase, the TC heuristic tries to move poorly-placed containers to stacks in which they are correctly-placed. This phase is called the *improvement phase* and the moves performed in this phase are called *improving moves*. Improving moves are equivalent to the "excellent moves" in Hottung & Tierney (2016). The containers with the largest time frames are first moved in this phase, because if these containers are correctly-placed in a stack, containers with a lower time frame can be correctly-placed on top of them. At first sight, it might seem wise to always perform the improving moves. However, in some scenarios, it is better not to perform improving moves and use the remaining pre-processing moves differently. Hence, the two variants of the improvement phase are: *performing improving* and *not performing improving* moves.

Phase 3: Selection

The number of pre-processing moves in the movement and improvement phase can be calculated for each combination of origin and destination stacks. There are in total S^2 possible combinations, but we only consider combinations of stacks for which fewer pre-processing moves were needed than the available pre-processing moves. For all these combinations, the remaining number of relocation moves are calculated using the rule-based estimation method presented in Zweers et al. (2020). This rule-based estimation method produces fast and rather accurate predictions for the number of relocation moves for a bay. Hence, we can estimate how much the objective function changes if the top container of the origin stack is placed correctly in the destination stack for each combination of origin and destination stack. However, even when for all feasible combinations of origin and destination stacks the number of expected relocation moves can be estimated, then it is still unclear which combination to select. It might be the case that one combination yields a strong decrease in the objective function and uses many pre-processing moves, while another combination only uses a few pre-processing moves, but has a smaller improvement.

In the selection phase of the TC heuristic, we distinguish two different selection methods: the *greedy* and the *ratio* selection method. In the greedy method, the combination that results in the largest decrease in the objective function is selected. On the other hand, in the ratio method, one divides the improvement in the objective function by the number of pre-processing moves needed to obtain the improvement per pre-processing move. As a result, the ratio selection method generally selects a solution in which a moderate improvement is obtained in a relatively small number of pre-processing moves. In contrast, in the greedy selection method, a solution is chosen that produces the

biggest improvement in a large number of pre-processing moves. Although the improvement made by the ratio section method is lower than the by the greedy selection method, it might be that many improving moves are possible for the bay that is selected. In that case, it is beneficial that still a rather large number of pre-processing moves is available for upcoming iterations of the improvement phase.

In both selection methods, there may be multiple combinations of origin and destination stack that result in the same improvement of the objective function. In that case, we simply select the leftmost origin stack and after that the leftmost destination stack. If no pre-processing moves can be found that result in a decrease in the expected number of relocation moves, the TC heuristic continues to the post-optimizing phase. Otherwise, it starts again in the movement phase.

Phase 4: Post-optimization

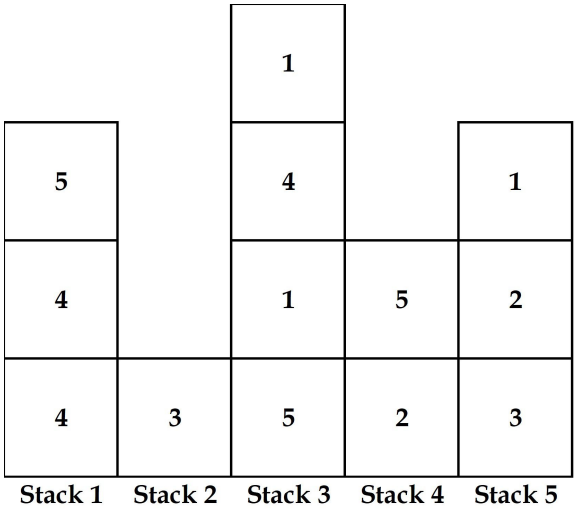
The three phases described above all have three two different options and thus there are in total eight different variants possible. Each of this eight variants terminates the moment that there is no feasible combination of origin and destination stacks that result in an improvement of the objective function. However, it could be that there are still some pre-processing moves available that are unused. The idea of the post-optimizing phase is to use these moves to change the bay without changing the estimate of the objective function. After the bay has changed, we again try the first three phases to see if an improvement can be made. It could be the case that a new top container can be easily placed correctly and with that improve the objective function.

In the post-optimization phase, we use the concept of *Good-Good* (GG) moves introduced by Tanaka & Tierney (2018) in the context of the CPMP. In a GG move, a well-placed container is moved to a stack in which it is still well-placed. It is likely that a GG move does not change the number of relocation moves in a bay. A GG move can be beneficial if after that GG move new improving moves are possible. The idea behind a GG move is to make the most improving moves possible. Hence, the container that is moved in a GG move is the container for which the difference between its time frame and the container underneath it is maximized. The destination stack is chosen such that the difference between the time frame of the top container and the time frame of the container that is moved is minimized. We apply a single GG move and then again the first three phases to improve the resulting bay. If no improvement is possible, another GG move is applied until the moment that either all pre-processing moves are performed or no GG moves are available.

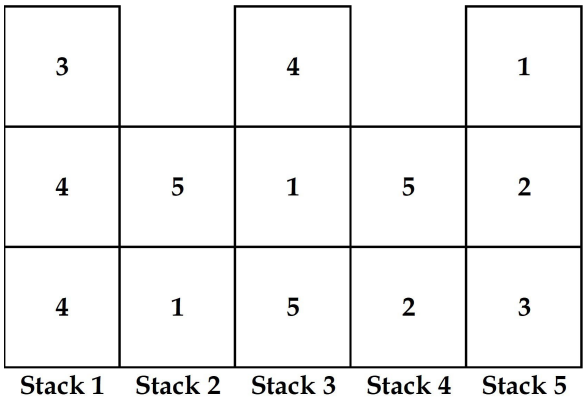
Running Example (continued)

Let us apply the TC heuristic to the bay of our running example in Figure 2. The estimate of the number of relocation moves for the initial bay of Figure 2 using the rule-based method is 6.2. In total, the TC heuristic produces eight solutions for this bay, but if the maximum number of pre-processing moves is three, then there are only two different solutions which are shown in Figure 4. All four variants in which the greedy selection method is used in the selection phase gives the bay in Figure 4(a), whereas the bay in Figure 4(b) is obtained using the ratio selection method.

Placing the top container of the fourth stack of the bay of Figure 2 at a correct position in the second stack needs three moves and then the bay in Figure 4(a) is obtained. In this case, the containers



(a) Bay of Figure 2 after performing three pre-processing moves using greedy selection.



(b) Bay of Figure 2 after performing three pre-processing moves using ratio selection.

Figure 4: Two possible outcomes of the TC heuristic after three pre-processing moves for the bay of Figure 2.

in stack 2 needs to be moved to a different stack, (which are a cleaning moves). It is not feasible to position these containers in the third stack, because that stack does already have the maximum number of containers. Moreover, these containers can neither be positioned in stack 4, because we would like to move the top container of the fourth stack. Hence, stacks 1 and 5 are the stacks to which they could be moved. The first container to be moved from stack 2 is the container with time frame 5. This container cannot be correctly placed in any of the two stacks and thus, it is positioned in the stack in stack 1 because in this stack it will be relocated in time interval 4, which is later than interval 2 of the fifth stack. Afterward, the container with time frame 1 can be positioned correctly in both stacks 1 and 5. As stack 1 has two poorly-placed containers and stack 5 does not contain any poorly-placed containers, it is moved to stack 5. In the final third move, the top container from stack 4 is moved to the second stack. For the bay of Figure 4(a), the estimate for the number of relocation moves is 4.8. This is the lowest that can be obtained by moving the top container of a stack in a correct position while using at most three pre-processing moves. Hence, that is why the greedy method selects this bay as the best bay.

In the first move of the ratio method, the top container of the fourth stack of the initial bay of Figure 2 is placed in the first stack. The estimated number of relocation moves for the bay that is obtained after this bay is 5.2 and thus the improvement per pre-processing move is exactly one. To compare that with the bay in Figure 4(a), the improvement per move in that bay is $\frac{6.2-4.8}{3} \approx 0.47$. After the first pre-processing move, only two pre-processing moves are left and the only top container that can be correctly positioned using one or two pre-processing moves is the top container of the middle stack. The improvement in the objective function is 0.5 if it is placed in stack 1, 4, or 5. Since stack 1 is the most left stack of these three stacks, the container is placed in that stack. Afterward, there is no top container that can be placed correctly in a stack with improving the objective function. Hence, the post-optimizing phase is entered. The top container of the first stack is now the container

with time frame 1 that has been moved in the second pre-processing move. This container is the only container for which we can apply a GG move. This container is still correctly positioned if it is moved to the fourth or fifth stack. The top container of the fourth stack has time frame 5, while the time frame of the fifth stack is only 2. Therefore, the container with time frame 1 is moved to the fifth stack, because then the difference with the top container is minimized. All in all, the bay of Figure 4(b) is obtained with the greedy selection method.

The optimal number of expected relocation moves for the bay in Figure 4(a) is $5\frac{1}{6}$ and for the bay in Figure 4(b), it is 5. Hence, for the bay of Figure 2 if three pre-processing moves are allowed, any variant of the TC heuristic that uses the ratio selection method produces a better solution than the variants that use the greedy selection method. However, for other bays or different maximum numbers of pre-processing moves the greedy selection method might produce better pre-processing moves than the ratio selection method.

4.2. Lower bound

In Section 4.3, a branch-and-bound algorithm will be developed for the SCRPCPP. For that reason, in this section, we present a lower bound for the SCRPCPP. For ease of explanation, we first describe in Section 4.2.1, the lower bound in the scenario in which each container has a unique time interval, i.e., the deterministic scenario. The lower bound for the stochastic setting is based upon the lower bound for this deterministic setting and is described in Section 4.2.2.

4.2.1. Deterministic CRP

In the CRP, it is easy to check whether a container needs to be relocated or not. If there is a container with a smaller time frame underneath the container under consideration, then it is relocated at least once, and otherwise, it is never relocated. Let us denote the lower bound for the number of relocation moves for container c by $lb(c)$, then $lb(c) = 1$ if $t(c) > u(c)$ and $lb(c) = 0$ otherwise. Hence, a lower bound for the number of relocation moves is to count all containers which need to be relocated at least once, which can be expressed as $\sum_{c\in\mathcal{C}} lb(c)$. So a lower bound for the number of relocation moves for the bay without any pre-processing moves can easily be calculated. The pre-processing moves are used to reduce the number of relocation moves, but each pre-processing move can reduce the lower bound for the relocation moves by at most one. Hence, a trivial lower bound for the CRP with pre-processing moves is $\max\{0, \sum_{c\in\mathcal{C}} lb(c) - \rho\}$.

However, for that lower bound it has to be possible to place each poorly-placed container in a correct position with only a single pre-processing move. There are two reasons why that might not be possible for a bay. The first reason is that there could be no stack with only containers with a higher time frame than the poorly-placed container. Secondly, it could be that the poorly-placed container has a well-placed container on top of it. In this case, the well-placed container needs to be moved before the poorly-placed container can be accessed. In case not all poorly-placed containers can become well-placed with a single pre-processing move, it might be possible to strengthen the lower bound. For that purpose, we divide the poorly-placed containers into two groups: the *one-move* and *multiple-moves* containers. The one-move containers can be moved to a correct position in a single pre-processing move and the multiple-moves containers need more than one pre-processing move to be placed correctly. If

the number of one-move containers is less than or equal to the number of pre-processing moves, then the lower bound for the CRP with pre-processing moves remains $\max\{0, \sum_{c\in\mathcal{C}} lb(c) - \rho\}$. Otherwise, at least one of the pre-processing moves is needed for a multiple-moves container and thus the lower bound is equal to: $\max\{0, \sum_{c\in\mathcal{C}} lb(c) - \rho + 1\}$. This improved lower bound is inspired by the lower bound of the CPMP given in Tanaka & Tierney (2018).

To check which containers are one-move containers, the bay needs to be investigated stack on stack level. If the top container of a stack is poorly-placed and can be placed correctly in another stack, it is a one-move container and the container underneath it can be investigated. Otherwise, the top container is a multiple-moves container and with that, all poorly-placed containers in that stack are multiple-moves containers. If the top container is a one-move container, then for the container underneath it we can also check if there is a stack in which it will be correctly-placed. If that is case, that container is also a one-move container and the next container of that stack is considered. Otherwise, the container is a multiple-moves container and the next stack is considered.

It is important to realize that no multiple-moves containers will become a one-move container after a one-move container has been moved in the pre-processing phase to a stack in which it is correctly-placed. If a one-move container has been moved to a stack in which it is correctly-placed, then the container with the minimum time frame of that stack is the one-move container that has just been moved. Hence, the minimum time frame of that stack has increased. Furthermore, the one-move container was poorly-placed in its origin stack and thus the minimum time frame of that stack has to be lower than the time frame of the one-move container. Consequently, after a movement of a one-move container, the lay-out has not improved for any multiple-moves container and no multiple-moves container will ever become a one-move container.

Running example (continued)

To illustrate the lower bound for the deterministic case, the bay of Figure 2 has been slightly adjusted. In Figure 5, the containers are positioned in the same way as in Figure 2, but each container has its own time interval. The order in which the containers in Figure 5 are retrieved is a possible retrieval order for the containers in Figure 2. In total, six containers need to be relocated at least once in the bay of Figure 5, namely the containers with time interval 2, 7, 8, 10, 11, and 13. Hence, the lower bound for the number of relocation moves for the bay in Figure 5 is six. Out of these six containers, the containers with time frame 2, 7, and 8 are one-move containers and the other containers are multiple-moves containers. When placed in stack 1, the container with time interval 7 does not need to be relocated anymore. Furthermore, the containers with time frame 2 can be correctly positioned in every other stack and if that container is moved, the container with time frame 8 is examined. This container can be placed in a correct position in stack 1. It is important to realize that in this specific example the order in which the containers are moved in the pre-processing phase is important. If container 7 is moved to the first stack before container 8, then container 8 cannot be correctly positioned in that stack. Nevertheless, as we are considering the lower bound we should focus on the best possible order.

If the number of pre-processing moves is three or less, then the three one-move containers can be moved and the lower bound equals six minus the number of pre-processing moves. For instance, the

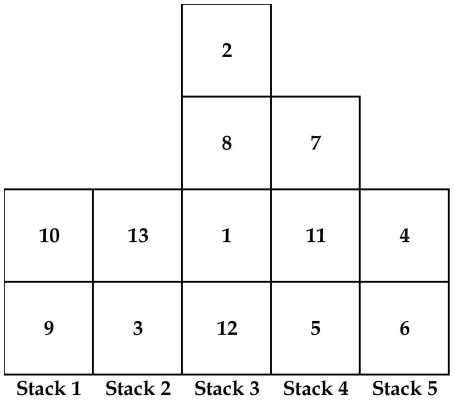


Figure 5: Deterministic variant of the bay in Figure 2 in which every container has a unique time interval.

lower bound for two pre-processing moves equals four. If there are more than three pre-processing moves, then there is at least one multiple-moves container that needs to be moved to further reduce the number of relocation moves. Therefore, the lower bound equals six minus the number of pre-processing moves plus one. For instance, the lower bound for five pre-processing moves is two. For seven and more pre-processing moves, the lower bound is zero.

4.2.2. Stochastic CRP

In the stochastic setting, both the lower bound for the number of expected relocation moves and the gain that can be achieved with pre-processing moves are different from the deterministic setting. However, the general idea of the lower bound is the same. A lower bound for the number of expected relocation moves is given in Galle et al. (2018b). This lower bound assigns to each container a probability that it needs to be relocated. The expected number of relocation moves is at least the sum of all these probabilities. Similarly to the deterministic case, this probability equals 0 or 1 for a container, if the minimum time frame of the containers underneath it is, respectively, strictly larger or strictly smaller than the time frame of the container. The relocation probability of a container lies strictly between 0 and 1 if its time frame is exactly the minimum time frame of the containers underneath it. To calculate the relocation probability, we use the value $m(c)$, which corresponds to the number of containers underneath container c with the same time frame as container c. If container c is retrieved before any of these $m(c)$ containers, container c is not relocated and otherwise it is relocated at least once. Since we made the assumption that the retrieval order is a uniform permutation, the probability that container c is picked up before the $m(c)$ containers with the same time frame is $\frac{1}{m(c)+1}$. Consequently, the probability that container c is relocated at least once is $\frac{m(c)}{m(c)+1}$, which is a lower bound for the expected number of relocation moves. The lower bound for the number of relocation moves for a bay is, just as in the deterministic case, the sum of the lower bounds for each container. The calculation of the lower bound on the number of expected relocation moves, corresponds to the first for-loop of Algorithm 1.

In the second for-loop of Algorithm 1, it is determined whether a container is a one-move or a multiple-moves container. Two aspects are different in the stochastic setting than in the deterministic setting. In the deterministic setting, it was only beneficial to place a container c in a stack in which

Algorithm 1: Lower bound for the objective function of the SCRPCPP.

Input: Bay B and maximum number of pre-processing moves ρ.
for $c \in \mathcal{C}$ **do**
  if $t(c) > u(c)$ **then**
    $lb(c) = 1$
  else
    if $t(c) = u(c)$ **then**
      $m(c) = |\{c' \in \mathcal{C} : s(c') = s(c) \wedge p(c') < p(c) \wedge t(c') = t(c) = u(c)\}|$
      $lb(c) = \frac{m(c)}{m(c)+1}$
    else
      $lb(c) = 0$
    end
  end
end
for $s \in \mathcal{S}$ **do**
  $t = n(s)$
  $I(s) = 0$
  while $t > 1$ **do**
    $c = C(t, s)$
    if $lb(c) > 0 \wedge t(c) \leq \max\{l(s') : s' \in \mathcal{S} \setminus s \wedge n(s') < T\}$ **then**
      $a = \min\{\frac{|\{c' \in \mathcal{C}: t(c)=t(c') \wedge s(c')=s'\}|}{|\{c' \in \mathcal{C}: t(c)=t(c') \wedge s(c')=s'\}|+1} : s' \in \mathcal{S} \setminus s \wedge t(c) \leq l(s') \wedge n(s') < T\}$
      $G(c) = \max\{lb(c) - a, 0\}$
      if $I(s) + G(c) > n(s) - t$ **then**
        $I(s) = I(s) + G(c)$
        $t = t - 1$
      else
        $t = 0$
      end
    else
      $t = 0$
    end
  end
end
if $\rho \leq \sum_{s \in \mathcal{S}} I(s)$ **then**
  $LB = \sum_{c \in \mathcal{C}} lb(c) - \rho$
else
  if $\rho \leq \sum_{s \in \mathcal{S}} I(s) + 1$ **then**
    $LB = \sum_{c \in \mathcal{C}} lb(c) - \sum_{s \in \mathcal{S}} I(s)$
  else
    $LB = \max\{\sum_{c \in \mathcal{C}} lb(c) - \rho + 1, 0\}$
  end
end
Output: LB.

the minimum time frame of all containers was higher than the time frame of c. However, in the stochastic setting it might also be beneficial to place container c in a stack which minimum time frame is the same as the time frame of container c . The value a in Algorithm 1 represents the relocation probability in the best stack for container c. Hence, the best improvement that can be made for moving container c in the pre-processing phase is given by $G(c)$, which is a value between 0 and 1. A second difference with the deterministic setting is how one should determine if the next container of a stack should be checked for being a one-move container. The value $I(s)$ is the improvement that can be made by placing one-move containers from stack s to other stacks. Only if $I(s) + G(c)$ is larger than the number of containers that needs to be moved before container c can be moved plus one, then container c belongs to the one-move containers. Otherwise, the improvement that can be made by moving container c is less than the number of containers in its stack, plus one, and if there is one extra move needed, then the container belongs to the multiple-moves containers.

After the improvement that can be made by only moving one-move containers is calculated, i.e., $\sum_{s \in \mathcal{S}} I(s)$, the final lower bound can be calculated. Similar to the deterministic case, if the number

of pre-processing moves is less than the improvement made by one-move containers, then the lower bound equals the lower bound for the relocation moves of the initial bay minus the number of pre-processing moves. If the number of pre-processing moves lies between the improvement made by the one-move containers and that improvement plus one, then the lower bound equals the lower bound for the number of relocation moves for the initial bay minus the improvement of the one-move containers. Finally, in all other scenarios the lower bound equals the lower bound for the relocation moves of the initial bay minus the number of pre-processing moves plus one.

Running example (continued)

Let us now compute the lower bound for the bay of the running example of Figure 2. The top container of the first and third stack has a relocation probability of $\frac{1}{2}$, because there are no containers with a strictly smaller time frame underneath them and only a single container with the same time frame. If the first stack consisted of three containers with time frame 4, then the top container would have had a relocation probability of $\frac{2}{3}$ and the middle container of $\frac{1}{2}$. For the containers with time frame 5 in stack 2, time frame 4 in stack 3, time frame 3 in stack 4, and time frame 5 in stack 4 at least one relocation move is needed and all other containers are correctly-placed. Hence, the lower bound for the relocation moves of the bay in Figure 2 is $4 + 2 \times \frac{1}{2} = 5$.

Although, the lower bound for the number of relocation moves for the top containers of the first two stacks is larger than zero, there is no stack in which they could be moved such that they are correctly-placed. Hence, the improvement that can be made for these two stacks is zero. The improvement for the fifth stack is also zero, because the top container is already correctly-placed. The top container of stack 4 can be placed correctly in the first stack, but there is no stack in which the container with time frame 5 underneath it can be placed such that the lower bound for the number of relocation moves for that container decreases. Therefore, the improvement for stack 4 is one. Finally, we consider the most interesting stack, which is stack 3. If the top container of that stack is placed in stack 1, 4, or 5, then the lower bound for the relocation moves of that container decreases from $\frac{1}{2}$ to zero, thus the gain for that container is $\frac{1}{2}$. The container below it with time frame 4 has a lower bound for the number of relocation moves of 1, which can be improved by placing the container in stack 1. If it is placed there, then the lower bound for the number of relocation moves of that container is $\frac{2}{3}$, thus the gain obtained by moving that is $\frac{1}{3}$. The total gain of these two containers is $\frac{5}{6}$ for which two containers need to be moved. However, if a multiple-moves container is moved, then the total gain with two moves could be one. Therefore, in the third stack, only the top container is considered as a one-move container and the total improvement of that stack is $\frac{1}{2}$.

All in all, the top containers of stacks 3 and 4 are one-move containers and moving these container into a correct position reduces the number of relocation moves by at least $1\frac{1}{2}$. Given that the lower bound for the number of relocation moves for the original bay is five, the lower bound with a single pre-processing move is four. With two pre-processing moves, the lower bound equals $3\frac{1}{2}$ and with three or more pre-processing moves, the lower bound is the maximum of zero and five minus the number of pre-processing moves plus one.

Algorithm 2: Branch-and-bound algorithm to find the optimal pre-processing moves.

Input: Bay B, relocation policy π and the maximum number of pre-processing moves ρ
$LB :=$ the lower bound from Algorithm 1.
$B' :=$ bay after pre-processing moves from TC.
$SOL := f(B', \pi)$
$Q := (B, \rho, LB)$
$I := \emptyset$
while $LB < SOL$ *and* $Q \neq \emptyset$ **do**
  Find the triplet $(B', d', LB') \in Q$ with the smallest d'. If there are multiple bays, choose the bay with the smallest value of LB'. In case there are still multiple bays left, choose the bay that was the earliest added to Q.
  $Q = Q \setminus \{(B', d', LB')\}$ and $I = I \cup \{(B', d', LB')\}$
  Set $VAL = f(B', \pi)$.
  if $VAL < SOL$ **then**
    $SOL = VAL$
  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 $LB'' < SOL$ *and* $\{(B, d, LB) \in Q \cup I : B = B'' \wedge d \geq d' - 1\} = \emptyset$. **then**
        $Q = Q \cup \{(B'', d' - 1, LB'')\}$
      end
    end
  end
end
Output: SOL

4.3. Branch-and-bound

In this section, a branch-and-bound algorithm is given for the SCRPCPP. The root of the branch-and-bound tree is the initial bay and for this bay, the TC heuristic can be applied to calculate an upper bound for the optimal solution. A lower bound for the optimal solution is derived by Algorithm 1. We expand the tree in depth-first way by constructing every possible bay that can be obtained with a single pre-processing move. Hence, the maximum depth of a tree is ρ. A node is only added to the tree if it is not already in the tree with fewer or the same number of pre-processing moves. Note that here we can exploit the fact that we can consider each permutation of a set of stacks as the same bay. If there are multiple nodes of the tree that have the same depth, the node with the lowest lower bound is selected and if that is also a tie, the node that was created the earliest is selected. For each bay that is selected, the expected number of relocation moves using relocation policy π is calculated. If this number is lower than the upper bound, the upper bound is adjusted. The complete branch-and-bound algorithm is given in Algorithm 2. This algorithm is similar to the branch-and-bound algorithm of Zweers et al. (2020), but there are two main differences. First, we have a tight bound on the depth of the tree, namely the number of pre-processing moves. Second, the best solutions are likely to be solutions with many pre-processing moves, whereas the problem in Zweers et al. (2020) usually has better solutions with fewer pre-processing moves. Hence, in Algorithm 2, the solutions with a higher depth are investigated first.

If one chooses the optimal relocation policy π^* and uses the exact evaluation of the number of expected relocation moves for a bay, then Algorithm 2 will produce the optimal solution. Nevertheless, for two reasons the branch-and-bound algorithm might take too much computational time for practical purposes to solve the SCRPCPP to optimality. The first reason is that the number of branches of the tree grows exponentially in the number of pre-processing moves. Second, the expected number of relocation moves has to be computed for each bay. The optimal PBFS algorithm, presented in

Galle et al. (2018b), is the best optimal algorithm for the relocation phase, but it has been shown that computing the optimal expected number of relocation moves for a single bay can already take more than one hour for larger instances. An alternative to computing the optimal expected number of relocation moves is using the rule-based estimation method of Zweers et al. (2020) to estimate the expected number of relocation moves in a bay. If that method is used, Algorithm 2 is not guaranteed to find an optimal solution, but it runs much faster. We refer to the branch-and-bound algorithm in which the optimal number of relocation moves is calculated as BB-O and use BB-H to indicate the variant of Algorithm 2 in which the rule-based estimation method is used to calculate the number of relocation moves for a bay.

5. Extension to multiple bays

In this section, we show how we can apply the methods described in the previous section to a situation in which we consider multiple bays. We first give in Section 5.1 the exact problem formulation for the extension to multiple bays. Afterward, in Section 5.2, we present a method to solve this problem.

5.1. Problem formulation for multiple bays

Some containers in a bay become correctly positioned in one or a few pre-processing moves, while for other containers possible many pre-processing moves might be needed to be in a position such that their expected number of relocation moves is lower. Hence, the improvement in the objective function per pre-processing move becomes smaller if more pre-processing moves are performed. Therefore, if the crane driver is idle for a longer time, then it could be better to visit multiple bays. Driving from one bay to the other is time-consuming, but if there are some containers that can be easily placed in a better position it might be worth moving the crane. Hence, in this section, we consider an extension of the SCRPCPP in which multiple bays are visited.

We assume that the bays are positioned in a line and are numbered from left to right as is shown in Figure 6. The crane in this figure is positioned above bay B_4, but at the beginning of the pre-processing phase it could be positioned above any bay. We call the bay above which the crane is positioned before the pre-processing phase the *starting bay*, which we denote by B_s. We do not require the crane to return to bay B_s at the end of the pre-processing phase, so any bay could be the last bay. In the extension to multiple bays, one needs to decide both the number of pre-processing moves to perform at each bay and a path that visits each bay for which at least one pre-processing move is performed. The total time needed to move the crane and perform the pre-processing times should again not exceed T. To reduce the problem's complexity, we do not allow the crane to take a container from one bay and place it in another bay.

The time needed to drive from bay B_i to bay B_j is given by t_{ij}. We restrict the travel times t_{ij} to the class of times that satisfy the following properties:

- $t_{ij} = t_{ji}$ for all B_i, B_j.
- $t_{ij} < t_{ik}$ for all $B_i < B_j < B_k$.

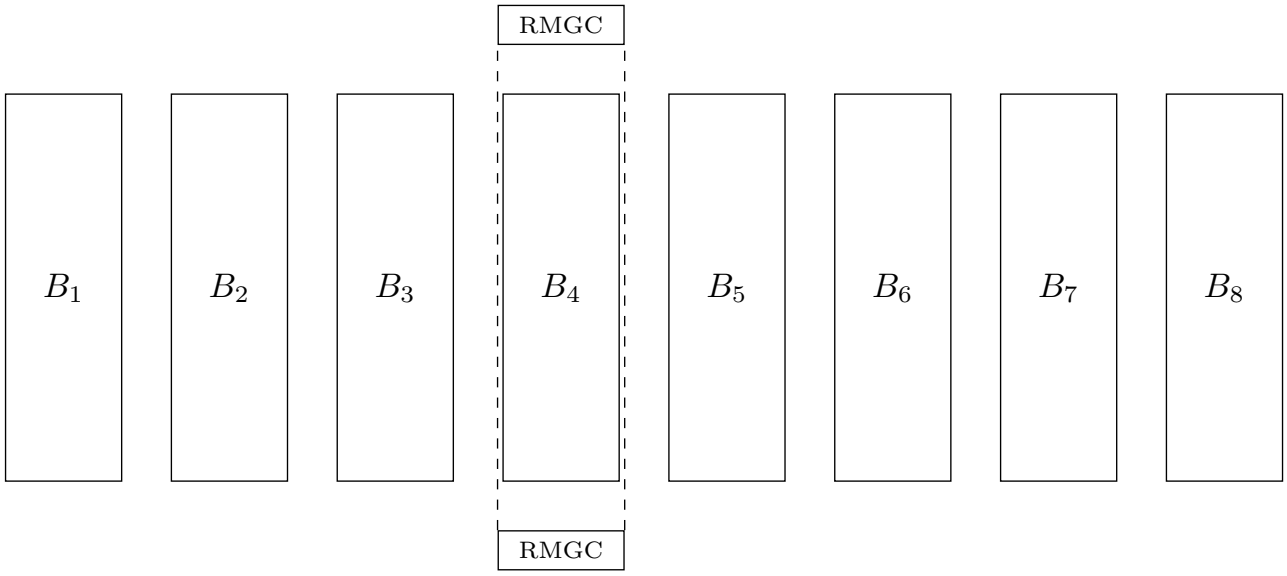


Figure 6: Illustration of the numbering of the bays (view from above).

- $t_{ij} + t_{jk} - t_{ik} = t_{lm} + t_{mn} - t_{ln}$ for all $B_i < B_j < B_k$ and $B_l < B_m < B_n$.

In the first property, we assume the travel times to be symmetric. The second property states that if one bay is closer than another bay, then the travel time is also shorter. The last equality implies that the cost of stopping at a bay is independent of the bay itself and the origin and destination of the bays that are visited before and after that bay. If one takes the layout of a container yard into account, as is illustrated in Figure 6, then the first two properties are realistic. With the last property, scenarios in which the crane drives at a constant speed and has no or a constant time needed to accelerate en decelerate are included. In the literature, these types of travel times have been assumed before (Lee & Lee, 2010; Lin et al., 2015). Nevertheless, we do not include the fact that a crane can accelerate stronger if it has to drive for a longer distance. Using these three properties for the travel time we can derive nice properties for the optimal path as are given in Lemma 1.

Lemma 1. *There exists an optimal path P with the following properties:*

1. *The bays in between the leftmost and rightmost bay are visited in either increasing or decreasing order.*
2. *The endpoint of the tour is either the leftmost or rightmost visited bay.*
3. *The bay visited directly after the start bay is either the leftmost or rightmost visited bay.*

The proof of Lemma 1 is given in Appendix A. There are two paths satisfying the properties of Lemma 1: one that ends at bay B_l and one that ends at bay B_r. The starting bay determines which of the two paths is optimal. The only difference in the length of these two paths is in the time needed to travel from the starting bay to the second visited bay. By the third property of Lemma 1, this bay is either B_r or B_l. The second property ensures that the other extreme bay that is visited is the last bay of the path. By the first property, we know that all bays in between bay B_r and B_l are visited in

the order they appear on the line. Since the travel times are symmetric, the costs of visiting the bays in between bays B_r and B_l are the same for starting in bay B_r or in bay B_l. As the only difference between the two paths is the time needed to go from the starting bay to the second bay, the optimal path visits first of B_r and B_l the bay that is closest to B_s. Then the crane travels to the other bay, B_l or B_r, and stops at every bay in between that needs to be visited. For example, consider the bay in Figure 6 and assume that all bays of this figure should be visited. As bay B_1 is closer to B_4 than B_8, the optimal path visits after bay B_4 first B_1 and then all bays in increasing order without visiting bay B_4 again.

Let X_{kl} be a binary decision variable indicating whether the crane drives from bay B_k to bay B_l or not. A bay is never visited more than once, because all moves that would be done in the second or later visits could also be done at the end of the first visit. Consequently, the constraint (2) for a single bay is replaced by the following constraint if there are multiple bays:

$$\sum_{k=1}^{m}\sum_{l=1}^{m} t_{kl}X_{kl} + \tau|\mathcal{P}| \leq T. \tag{3}$$

5.2. Solution method for multiple bays

In Sections 4.1 and 4.3, we have given an optimal algorithm and two heuristic methods to find pre-processing moves for a single bay. We use these method in this section to derive a solution method for the extension to multiple bays. The idea of the method for multiple bays is to calculate, for each bay and number of pre-processing moves, the improvement that can be made by performing that number of pre-processing moves in that bay. For a single bay k and any number of λ_k pre-processing moves, the (optimal) pre-processing moves can be determined using the methods from Sections 4.1 and 4.3. Let us define $S(B_k, \lambda_k)$ as (an estimation of) the expected number of relocation moves for bay B_k after λ_k pre-processing moves have been performed. Moreover, for a specific bay B_k we can also determine the maximum number of pre-processing moves that will be done for that bay, which we denote by m_k. First of all, we know that m_k cannot exceed ρ because of the time limit. Second, if with λ_k pre-processing moves $S(B_k, \lambda_k) = 0$, then it is useless to perform more than λ_k pre-processing moves. In other words, $m_k := \min\{\rho, \arg\min_\mu\{S(B_k, \mu) = 0\}\}$. Hence, the maximum number of pre-processing moves that can be performed in a bay is different per bay. Consequently, the values that λ_k can take is different for each bay k. For each bay B_k we can calculate the value of $S(B_k, \lambda_k)$ for $\lambda_k = 0, 1, \ldots, m_k$. If the optimal pre-processing moves are performed and the value of $S(B_k, \lambda_k)$ is the optimal number of relocation moves, then $S(B_k, \lambda_k) \geq S(B_k, \lambda_k + 1)$. However, if either the pre-processing moves or the estimation of the relocation moves are not optimal, the inequality does not always hold. Nevertheless, if the estimates $S(B_k, \lambda_k)$ are calculated in an iterative order starting at $\lambda_k = 0$, then one can set the value $S(B_k, \lambda_k + 1)$ equal to $S(B_k, \lambda_k)$ if $S(B_k, \lambda_k + 1) > S(B_k, \lambda_k)$. Hence, we know that for each bay B_k it should hold that $S(B_k, 0) \geq S(B_k, 1) \geq \ldots, S(B_k, m_k)$.

Using the values for $S(B_k, \lambda_k)$, we can formulate an ILP to decide upon the number of pre-processing moves for each bay and the route of the crane. Let $Y_{k\lambda_k}$ be a binary variable that equals one if λ_k pre-processing moves are performed in bay B_k. Furthermore, let X_{jk} be a binary variable indicating whether the crane travels from bay B_j to bay B_k. In Lemma 1, it has been shown that,

given a set of bays that needs to be visited, an optimal path has some structural properties. Therefore, we know that either the leftmost bay (B_l) or rightmost bay (B_r) is the first bay to be visited after the initial bay and the other of the two is the last bay that is visited. We give the starting bay an index of 1. We could number the other bays either from left to right or from right to left. If we use the ILP below to solve both numberings then the solution with the lowest objective function is the optimal solution. All in all, the SCRPCPP for multiple bays can be formulated as the following ILP:

$$\min \quad \sum_{k=1}^{b} \sum_{\lambda_k=0}^{m_k} S(B_k, \lambda_k) Y_{k\lambda_k} \tag{4}$$

subject to:

$$\sum_{\lambda_k=0}^{m_k} Y_{k\lambda_k} \leq 1 \qquad k = 1, \ldots, b \tag{5}$$

$$\sum_{\lambda_k=1}^{m_k} Y_{k\lambda_k} \leq \sum_{l=1}^{b} X_{lk} \qquad k = 2, \ldots, b \tag{6}$$

$$\sum_{o=1}^{b} X_{lo} \leq \sum_{k=1}^{b} X_{kl} \qquad l = 2, \ldots, b \tag{7}$$

$$\sum_{k=1}^{b} \sum_{l=1}^{b} t_{kl} X_{kl} + \sum_{k=1}^{b} \sum_{\lambda_k=0}^{m_k} \tau k Y_{k\lambda_k} \leq T \tag{8}$$

$$X_{ij} = 0 \qquad i = 1, \ldots, b \quad j = 1, \ldots, i \tag{9}$$

$$X_{ij} \in \{0, 1\} \qquad i = 1, \ldots, b \quad j = 1, \ldots, b \tag{10}$$

$$Y_{k\lambda_k} \in \{0, 1\} \qquad k = 1, \ldots, b \quad \lambda_k = 0, \ldots, m_k. \tag{11}$$

In the objective function in equation (4) the sum of $S(B_k, \lambda_k)$ over all bays and all possible pre-processing moves in each bay is taken. In constraint (5), it is forced that for each bay only a single number of pre-processing moves can be performed. The X- and Y-variables are coupled in constraint (6) by forcing all $Y_{k\lambda_k}$ to be zero if the crane does not visit bay k. Note that this does not hold for bay 1, because the crane starts in that bay. Constraint (7) ensures that the crane can only leave a bay if it has arrived at that specific bay. Again, this does not apply to the starting bay. The first sum in constraint (8) represents the total travel time of the crane between the bays and the second sum of this constraint gives the total time the crane spends on moving containers inside the bays. Thus, in constraint (8) the total time the crane uses in the pre-processing phase is limited to T. In constraint (9), the properties of Lemma 1 are exploited with the fact that there exists an optimal tour in which the bays are visited in the way they are numbered. Finally, constraints (10) and (11) ensure that all variables are binary.

In the ILP above, the fact that the maximum number of pre-processing moves in a bay is different per bay is used. One could also decide to refrain from calculating m_k for each bay k and decide to set m_k equal to ρ. This has the advantage that notation can be simplified because λ_k and m_k do not depend on k anymore. However, this comes at the costs of having more Y-variables and thus, we have

1	Correct, No improvement, Ratio
2	Correct, No improvement, Greedy
3	Correct, Improvement, Ratio
4	Correct, Improvement, Greedy
5	Semi-correct, No improvement, Ratio
6	Semi-correct, No improvement, Greedy
7	Semi-correct, Improvement, Ratio
8	Semi-correct, Improvement, Greedy

Table 2: Numbering of the TC heuristic.

decided not to do so.

6. Numerical Results

In this section, numerical experiments are conducted to compare the three solution methods for a single bay presented in the previous section: the TC heuristic, the BB-H method, and the BB-O method. Moreover, the gain that can be obtained by investigating multiple bays is investigated. First, we compare in Section 6.1, the eight different variants of the TC heuristic. The results of the TC heuristic are compared with the branch-and-bound methods in Section 6.2. Finally, we compare the effects of extending the problem to multiple bays in Section 6.3. All the numerical experiments are done using the set of instances for the SCRP of Ku & Arthanhari (2016). This set contains bays with five up to ten stacks and a maximum stack height of three up to six containers. Furthermore, the number of containers inside a bay is either 50% or 67% of all slots in the bay. This percentage is referred to as the *fill rate*. For each combination of these parameters, 30 instances are in the set of instances.

We decided to restrict the experiments to the instances of Ku & Arthanhari (2016) with five and ten stacks to investigate the consequences of different number of stacks. Furthermore, we only focus on the instances with a fill rate of 67%, because they have more containers per stack. These instances have, in general, a higher number of expected relocation moves than the instances with a fill rate of 50%, because these bays contain more containers. Another consequence of the higher number of containers is that more pre-processing moves are needed to reduce the relocation moves. Hence, we can assume that the instances with a fill rate of 67% are more difficult than the instances with a fill rate of 50%. Finally, we solve every instance for three different number of pre-processing moves. We let the number of pre-processing moves depend on the number of containers in a bay. We set the number of pre-processing moves to 25%, 50%, and 75% of the total number of containers in a bay. It is important to note that if the number of pre-processing moves equals 25% of the total number of containers, then some instances might already be reshuffled such that no relocation moves are needed. In that case, performing more pre-processing moves is obviously not useful. However, there will also be instances for which still relocation moves will be needed if the number of pre-processing moves equals 75% of the total number of containers.

6.1. TC heuristic

In this section, the eight different variants of the TC heuristic, as discussed in Section 4.1, are compared with each other. In order to refer easily to the variants of the TC heuristic, we have given

Instances	# heuristics	1	2	3	4	5	6	7
All	% diff 8 heur.	16,05%	5,16%	1,75%	0,93%	0,40%	0,13%	0,01%
	Best heuristics	7	7,8	1,7,8	3,5,7,8	3,5,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$\rho = 0.25C$	% diff 8 heur.	5,08%	1,52%	0,39%	0,17%	0,05%	0,01%	0,01%
	Best heuristics	8	5,8	5,7,8	2,5,7,8	2,3,5,7,8	3,4,5,6,7,8	1,3,4,5,6,7,8
$\rho = 0.5C$	% diff 8 heur.	20,92%	6,42%	1,62%	0,65%	0,13%	0,05%	0,01%
	Best heuristics	7	7,8	1,7,8	1,6,7,8	1,3,6,7,8	1,3,4,6,7,8	1,3,4,5,6,7,8
$\rho = 0.75C$	% diff 8 heur.	40,09%	16,82%	6,18%	3,26%	1,20%	0,17%	0,03%
	Best heuristics	3	3,8	3,5,8	1,3,7,8	1,3,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$H = 3$	% diff 8 heur.	10,45%	2,55%	1,09%	0,55%	0,09%	0,00%	0,00%
	Best heuristics	8	7,8	1,7,8	3,5,7,8	2,3,5,7,8	2,3,5,6,7,8	2,3,4,5,6,7,8
$H = 4$	% diff 8 heur.	13,16%	4,26%	2,09%	1,03%	0,53%	0,06%	0,03%
	Best heuristics	8	3,8	3,6,8	3,5,6,8	3,5,6,7,8	3,4,5,6,7,8	1,3,4,5,6,7,8
$H = 5$	% diff 8 heur.	13,93%	3,81%	0,94%	0,41%	0,13%	0,02%	0,00%
	Best heuristics	8	5,8	1,7,8	1,3,7,8	1,3,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$H = 6$	% diff 8 heur.	13,01%	5,44%	1,97%	0,87%	0,34%	0,11%	0,02%
	Best heuristics	7	7,8	1,7,8	3,5,7,8	3,5,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$S = 5$	% diff 8 heur.	13,47%	4,51%	1,76%	0,71%	0,41%	0,13%	0,02%
	Best heur.	8	7,8	1,7,8	1,6,7,8	1,5,6,7,8	1,3,5,6,7,8	1,3,4,5,6,7,8
$S = 10$	% diff 8 heuristics	16,65%	5,78%	1,54%	0,64%	0,26%	0,13%	0,01%
	Best heuristics	7	7,8	3,5,8	3,5,7,8	3,5,6,7,8	3,4,5,6,7,8	1,3,4,5,6,7,8

Table 3: Comparison of different variants of the TC heuristic for different subsets of the instances.

each variant of the TC heuristic a number which is given in Table 2. In Table 3, the effect of using eight variants of the TC heuristic is investigated. For each variant and instance, we calculated the number of expected relocation moves of the bay after the pre-processing phase. Calculating the optimal number of expected relocation moves for some of the instances takes already more than an hour. Hence, we decided to estimate the expected number of relocation moves by solving 10,000 realizations of the retrieval order using the EM heuristic. On the rows of Table 3 are different subsets of the instances and on the columns are the number of variants of the TC heuristic that are used. Both the best combination of variants and the percentage difference with the objective function if all eight variants were used are given. For example, if all instances as described above, are considered, then heuristic 7 is the best if only a single variant is allowed. The percentage difference in objective function between the solution of heuristic 7 and the solution using all eight variants is 16.05%.

There are three main conclusions to be drawn from Table 3. The first is that the solution of the TC heuristic greatly improves in the beginning by adding more variants, but that the contribution of the sixth, seventh and eight best variants is marginal. For example, if the number of pre-processing moves is 75% of the total number of containers in the bay ($\rho = 0.75C$), then the solution of the best heuristic, heuristic 3, is more than 40% worse than the solution including all eight variants. However, when allowing for three variants, the solution is already only 6.18% off from the solution using eight variants. A second conclusion is that the number of stacks or the maximum height of a stack does not really influence how many heuristics are needed to obtain a solution that is close to the best possible solution of the TC heuristic. However, the number of pre-processing moves has a big impact on the number of heuristics needed. If the number of pre-processing moves is only 25% of the total number of containers then the variants have not got many options to vary in the suggested pre-processing moves. In case the number of pre-processing moves is 75% of the total number of containers, then the number of possible different solutions is much bigger. The third main conclusion is that there is no variant of the TC heuristic that outperforms the others in all subsets of the instances. Heuristic 7 is the best for all instances, but that is partially caused by the fact that it performs well on the instances with

more relocation moves, namely $H = 6$ and $S = 10$. Although heuristic 7 is the best for $S = 10$ if only a single heuristic is selected, the best three heuristics do not include 7. Another observation is that if two heuristics can be chosen, then heuristic 8 is always one of the two.

All in all, it is beneficial to include multiple variants in the TC heuristic and although certain heuristics give better results than others, no variant is outperforming all other heuristics for every type of instance. In the remainder of the paper, all eight variants of the TC heuristic will be applied to obtain the best possible results. However, if time does not allow to run all eight variants, only including the best four or five heuristics is probably the best option.

6.2. Branch-and-bound methods

In this section, the performance of the TC heuristic is compared with the two branch-and-bound methods introduced in Section 4.3: BB-H and BB-O. Again, we only focus on the set of instances of Ku & Arthanhari (2016) with five and ten stacks and a fill rate of 67%. In Table 4, the three methods TC, BB-H, and BB-O are compared for the instance with five stacks. In Table 5, exactly the same is done but for the instances with ten stacks. Similar to the previous section, we set the maximum number of pre-processing moves to 25%, 50%, and 75% of the total number of containers in a bay. Moreover, we also include a column with $\rho = 0$ to investigate the improvement that is made in the pre-processing phase. For each combination of stack height and maximum number of pre-processing moves there are thirty instances and for each instance the maximum running time for the pre-processing phase is set to one hour. We report in Tables 4 and 5, the average running time per instance. In calculating the average running time, we include the instances for which the pre-processing phase was stopped after one hour. For the TC heuristic, all eight heuristics were run, which all have about the same running time. The running time of the TC heuristic reported in Tables 4 and 5 could be decreased by solving only a subset of the eight variants, but in that case, the solution quality also decreases as we have seen in Section 6.1. Furthermore, we show in the row 'Solved', for how many of the thirty instance the pre-processing phase terminated within one hour. This number is especially important for the BB-O because if this method solves an instance within one hour then the solution is optimal. Therefore, if all thirty instances are solved, then the sum of objective values is optimal. In order to calculate the objective function of the SCRPCPP, we need to calculate the remaining expected number of relocation moves after the pre-processing phase. We calculate these expected relocation moves in two different ways: we use both the optimal algorithm and the EM heuristic of Galle et al. (2018a). We set the maximum running time for the relocation phase also to one hour and in that case, the relocation phase cannot be solved to optimality for the larger instances. Therefore, we use the EM heuristic to also obtain an estimate for the objective function for these instances. For the EM heuristic we use 10,000 simulations to calculate the average number of relocation moves for an instance.

The first conclusion from Table 4 and 5 is that the maximum height of the stacks has the biggest influence on the running time for all methods. For example, the number of containers in a bay is the same for $S = 5$ and $H = 6$, and $S = 10$ and $H = 3$. However, in the latter, the average running time of the optimal method in only a few seconds, whereas in the former, the optimal method could only solve six instances to optimality within one hour. Furthermore, the running times of the BB-O and BB-H methods also increase significantly with the maximum number of pre-processing moves. For some

		$\rho = 0$	$\rho = 0.25C$			$\rho = 0.5C$			$\rho = 0.75C$		
			BB-O	BB-H	TC	BB-O	BB-H	TC	BB-O	BB-H	TC
$H = 3$	Opt obj value	92.3	49.9	50.6	50.9	8.5	8.5	11.5	2	2	3
	EM obj value	92.3	49.9	50.6	50.9	8.5	8.5	11.5	2	2	3
	Avg run time (s.)	-	0.9	0.9	0.9	0.9	0.9	0.9	1.2	0.9	0.9
	Solved	-	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30
$H = 4$	Opt obj value	167.4	95.2	97.5	99.5	39.6	41.3	45.3	7.0	7.2	25.3
	EM obj value	169.4	95.3	97.6	99.6	39.6	41.6	45.3	7.0	7.2	25.3
	Avg run time (s.)	-	3.7	1.5	1.4	3.2	1.5	1.4	53.7	7.9	1.4
	Solved	-	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30	30/30
$H = 5$	Opt obj value	-	187.0	191.4	193.1	101.2	95.7	115.6	58.2	35.3	72.6
	EM obj value	314.4	191.2	202.0	198.6	102.8	98.7	117.1	58.8	35.3	72.8
	Avg run time (s.)	-	1200.6	2.2	2.2	1441.2	6.2	2.4	2864.7	1210.4	2.3
	Solved	-	22/30	30/30	30/30	22/30	30/30	30/30	8/30	22/30	30/30
$H = 6$	Opt obj value	-	-	-	-	199.9	145.5	210.1	136.7	92.8	140.7
	EM obj value	439.5	286.6	292.8	312.6	204.8	149.4	215.4	139.3	95.3	143.3
	Avg run time (s.)	-	3018.0	15.1	2.8	3491.0	674.6	2.8	3483.8	3371.1	3.0
	Solved	-	6/30	30/30	30/30	2/30	27/30	30/30	1/30	2/30	30/30

Table 4: Sum of objective function and average running times of the BB-O, BB-H and TC solution methods for $S = 5$ and fill rate of 67% for different stack heights and number of pre-processing moves.

		$\rho = 0$	$\rho = 0.25C$			$\rho = 0.5C$			$\rho = 0.75C$		
			BB-O	BB-H	TC	BB-O	BB-H	TC	BB-O	BB-H	TC
$H = 3$	Opt obj value	158.2	40.5	40.7	50.0	1.0	1.0	2.5	0	0	0
	EM obj value	158.4	40.5	40.7	50.0	1.0	1.0	2.5	0	0	0
	Avg run time (s.)	-	3.3	2.7	2.6	150.5	144.0	3.5	4.1	4.1	4.1
	Solved	-	30/30	30/30	30/30	29/30	29/30	30/30	30/30	30/30	30/30
$H = 4$	Opt obj value	-	137.3	138.8	157.3	33.5	26.5	43.0	3.5	2.5	3.5
	EM obj value	308.8	137.5	139.3	157.6	33.5	26.5	43.0	3.5	2.5	3.5
	Avg run time (s.)	-	1028.3	13.2	4.8	2708.4	2061.7	5.8	487.9	487.2	6.9
	Solved	-	25/30	30/30	30/30	15/30	9/30	30/30	26/30	26/30	30/30
$H = 5$	Opt obj value	-	-	-	-	125.7	104.0	128.2	48.6	47.6	48.6
	EM obj value	485.7	271.9	253.4	284.5	125.7	104.0	128.2	48.6	47.6	48.6
	Avg run time (s.)	-	3094.4	331.8	7.0	3600.0	3564.3	9.4	3001.9	3001.7	10.7
	Solved	-	7/30	28/30	30/30	0/30	1/30	30/30	5/30	5/30	30/30
$H = 6$	Opt obj value	-	-	-	-	-	-	-	-	-	-
	EM obj value	696.0	425.5	394.8	432.5	230.6	198.4	230.6	111.3	109.8	111.3
	Avg run time (s.)	-	3600.0	1252.4	9.0	3600.0	3600.0	11.8	3600.0	3600.0	13.0
	Solved	-	0/30	22/30	30/30	0/30	0/30	30/30	0/30	0/30	30/30

Table 5: Sum of objective function and average running times of the BB-O, BB-H and TC solution methods for $S = 10$ and fill rate of 67% for different stack heights and number of pre-processing moves.

parameter settings, for example $S = 5$, $H = 5$, and $\rho = 0.5C$ and $S = 10$, $H = 4$, and $\rho = 0.25C$, the BB-H method can still solve the instances within seconds, whereas the BB-O method cannot find the optimal solution for every instance within one hour. There are other parameter settings, especially when $S = 10$, for which also the BB-H method cannot solve all instances within one hour. Consequently, for instances in which the maximum stack size or the number of pre-processing moves is large, only the running time of the TC heuristic is small enough to be applied in practice.

We also see that in case the optimal expect relocation moves could be calculated within one hour, the objective value approximated by the EM heuristic is always within 5% of the optimal objective function. If the number of containers in a bay is large, then it is not possible to calculate the optimal expected number of relocation. More interestingly, this is also the case when fewer pre-processing moves have been applied. For example, for $S = 5$ and $H = 6$, the optimal relocation moves could only be calculated for $\rho = 0.5C$ and $\rho = 0.75C$, but not for $\rho = 0$ and $\rho = 0.25C$. Hence, we can conclude that the relocation phase is simplified if more pre-processing moves are applied.

It is important to note that if the BB-O method does not solve all instances within one hour, it

is often outperformed by the BB-H method. At first, this might be counterintuitive, but it can be explained by the fact that the BB-H method needs less time to investigate the quality of a node in the branch-and-bound tree. In the BB-O method, to investigate the quality of a node, the optimal number of expected relocation moves needs to be calculated, whereas in the BB-H method, only an estimation of the number of relocation moves is used. As a result, the BB-H method visits more nodes in the branch-and-bound tree than the BB-O method and finds better solutions.

If one wants to compute the optimality gap of the BB-H or TC method, it is important to realize that this can be in two different ways that give completely different results. For instance, for $S = 5$, $H = 4$, and $\rho = 0.75C$, one could compare the values 7.0 and 25.3 and argue that the optimality gap of the TC heuristic is more than 300%. However, one could also look at the reduction of the objective function, compared with $\rho = 0$. In this situation, the optimality gap of the TC heuristic is only about 10%. Comparing the objective function for different values of ρ, we can also conclude that the first pre-processing moves yield the largest reduction in the number of relocation moves. For instance, if we take a look at the difference of the objective function between $\rho = 0$ and $\rho = 0.75C$, we see that about 40% to 50% of this improvement is already made for $\rho = 0.25C$. Whereas the improvement made between $\rho = 0.5C$ and $\rho = 0.75C$ is only about 10% to 15%. The observation that the first pre-processing moves yield a bigger reduction in the objective function than later pre-processing moves supports the idea to investigate multiple bays.

6.3. Multiple bays

If the idea of pre-processing is extended to multiple bays, the time units that are used are important. We use as closely as possible the time units that were used in previous works in the literature (Lee & Lee, 2010; Lin et al., 2015). We assume that the time to move a container inside a bay is 30 seconds. Furthermore, the time to accelerate and decelerate the crane is set to be 40 seconds and the time to move the crane a single bay is 3.5 seconds per bay. For example, going with the crane to an adjacent bay takes 43.5 seconds and to a bay that is separated by two other bays in between takes 50.5 seconds. As we are the first to study any variant of the SCRP in a multiple bay setting, no benchmark instances are available. We have decided to use the same instances from Ku & Arthanhari (2016) as for the single-bay case. For each parameter setting thirty single-bay instances are available. An instance for multiple bays consists out of twenty randomly selected instances of these thirty instances. We investigate the instances with five and ten stacks and with a maximum height of five containers. For both of these parameter settings, we have created thirty instances consisting of twenty bays. Furthermore, we compare the effect of the starting position of the crane. The crane can be positioned at the beginning or in the middle of the yard. If the crane is positioned at the beginning of the yard, fewer bays are located nearby than if the crane is located in the center of the yard. Finally, we use four different variants of T, namely $T = 300$, $T = 600$, $T = 900$ and $T = 1800$, which corresponds to five, ten, fifteen and thirty minutes.

In Table 6, the results of the ILP of Section 5.2 are shown for the parameter set described above. For the pre-processing phase, the TC heuristic is used and to estimate the number of relocation moves we have used the EM heuristic based on 10,000 simulations. We have chosen to do so because each bay has to be solved with possibly a large number of pre-processing moves. One conclusion that can

		$S = 5$		$S = 10$	
	Start crane	Begin	Middle	Begin	Middle
$T = 300$	# pre-proc. moves	7.80	7.63	7.73	7.70
	# bays visited	1.43	1.27	1.57	1.67
	Reduction # reloc. moves	9.52	9.45	9.71	10.09
$T = 600$	# pre-proc. moves	15.73	15.70	15.77	15.70
	# bays visited	2.93	2.70	2.60	2.83
	Reduction # reloc. moves	19.05	18.75	19.37	19.62
$T = 900$	# pre-proc. moves	24.40	24.10	23.87	24.37
	# bays visited	3.83	3.70	3.80	3.83
	Reduction # reloc. moves	28.11	27.51	28.67	28.97
$T = 1800$	# pre-proc. moves	48.30	48.43	49.70	49.83
	# bays visited	8.03	7.77	7.00	6.77
	Reduction # reloc. moves	53.26	52.49	54.76	54.53

Table 6: The average number of pre-processing moves, number of bays visited and the improvement made in the objective function for different values of T, S and the starting position of the crane.

be drawn from Table 6 is that the results are rather similar for instances with five and ten stacks per bay. However, with ten stacks the number of relocation moves can be reduced slightly more. There are two possible explanations of why the objective function can improve more for instances with ten stacks than with five stacks. The first is that every initial bay has on average a larger number of relocation moves and thus performing more pre-processing moves in a single bay is likely to give a greater improvement in the objective function. This effect is the most clearly seen for $T = 1800$ for which for $S = 10$ more pre-processing moves per bay are performed than for $S = 5$. A second explanation is that in a bay with ten stacks, it happens more often that a poorly-placed container can be placed correctly using a single pre-processing move than in a bay with five stacks. This is caused by the fact that there are simply more potential stacks in a bay with ten stacks than in a bay with five stacks. Consequently, if $S = 5$ it is often the case that no pre-processing moves are done in the starting bay, whereas if $S = 10$ usually at least a few pre-processing moves are performed in the starting bay. The advantage of performing pre-processing moves in the starting bay is that no travel time is needed. Hence, if the time is limited, for instance for $T = 300$, we see that the instances with ten stacks have more bays visited.

If the crane is positioned at the center of the yard, then the other bays are on average closer than if the crane starts at the beginning of the yard. This might be beneficial if only one or two bays could be visited. Nevertheless, if either the leftmost or rightmost bay is close to the end at which the crane is positioned, then starting at the beginning of the yard is more beneficial, because the crane does not need to travel that far in the beginning. If the available time is larger and more bays are visited, it is likely that the leftmost and rightmost visited bays are closer to the end of the yards. Hence, we see that for $T = 1800$ the improvement is larger if the crane is positioned at the beginning of the yard. Obviously, these results heavily depend on the values for t and τ.

Furthermore, as expected, the improvement per pre-processing move decreases if T increases, because if T is small only the very best pre-processing moves are performed, whereas if T gets larger then also less profitable pre-processing moves are performed. However, even for $T = 1800$, the improvement per pre-processing move is still larger than one, meaning that one pre-processing move reduces, on average, the number of relocation moves with more than one. If one does not allow the crane to move and let it only be positioned above a single bay, then this improvement is much lower. For instance, if

$T = 900$, the crane could perform thirty pre-processing moves, which yields an average improvement of 10.22 for $S = 5$ and when $S = 10$ of 15.67. However, even when the available time is only 300, already a substantial improvement can be made. In that case, ten pre-processing moves can be performed for a single bay, which yields an improvement of 7.05 and 8.34 for, respectively, $S = 5$ and $S = 10$.

A final remark to be made concerns the running time of the extension to multiple bays. The ILP (4)-(11) runs in less than a second for the instances we have considered. Hence, the only time-consuming part is to calculate the values of $S(B_k, \lambda_k)$ for each bay B_k and number of pre-processing moves λ_k. Here a trade-off has to be made between the solution quality and the running time. If one uses a single variant of the TC heuristic for the pre-processing phase and estimates the number of relocation moves by the rule-based method, then even for the largest instances calculating $S(B_k, \lambda_k)$ takes at most two seconds. Nevertheless, for a better solution quality more computing time is needed. As the values of $S(B_k, \lambda_k)$ do not depend on the other bays it could be possible in practice to calculate them offline. Only if a container arrives or leaves in a bay, the values of $S(B_k, \lambda_k)$ have to be updated.

7. Conclusion

In this paper, a new optimization problem, the SCRPCPP, faced by an inland container terminal in the port of Amsterdam, has been introduced. The problem focuses on which pre-processing moves to perform to reduce the expected number of relocation moves as much as possible. We have developed a branch-and-bound algorithm to solve this problem to optimality. However, for larger instances, the running time of this method is too large to be used in practice. Therefore, we have also developed two heuristics. The first heuristic, the TC heuristic is fast, but the gap between its solution and the optimal solution is for more difficult instances large. The BB-H heuristic is based on the optimal branch-and-bound algorithm, but as it does not calculate the optimal number of relocation moves for a bay, its solutions are not necessarily optimal. However, its running time is also significantly lower than the optimal algorithm. We have also developed a method for the pre-processing phase of a complete yard consisting out of multiple bays. Using multiple bays, one can make a large improvement per pre-processing move even when a large amount of time is given for the pre-processing phase.

A direction for further research could be to improve the formulation for multiple bays. In the current method, for each combination of bays and possible number of pre-processing moves one has to calculate the improvement, which could be time-consuming. A fast method to decide upon the allocation of the number of pre-processing moves to a bay based on the initial layout could be an improvement. Nevertheless, in allocating the pre-processing moves, one should also take into account the position of the bay in the yard. Furthermore, this new method might allow for moving a container from one bay to the other.

Both the TC and BB-H heuristic rely on the rule-based estimation method for estimating the number of relocation in a bay. If a better estimation method is found, then the performance of both methods can be improved. We expect that if sophisticated machine learning algorithms are applied then better predictions might be obtained than by the rule-based estimation method. Furthermore, at the moment the BB-O and BB-H heuristic do not use advanced branching decisions, it might also be worth to investigate if their performance could improve if better branching decisions were used.

Finally, one could speed-up the BB-H heuristic by only constructing a partial tree or using a beam search approach to focus only on the promising parts of the tree. It has to be investigated if the solution quality of such type of method is much lower than the current BB-H heuristic.

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., & Lee, C.-Y. (2014). A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics*, *61*, 101–118.

Bortfeld, A., & Forster, F. (2012). A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research*, *217*, 531–540.

Carlo, H., Vis, I., & Roodbergen, K. (2014). Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, *235*, 412–430.

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

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.

Casey, B., & Kozan, E. (2012). Optimising container storage processes at multimodal terminals. *The Journal of the Operational Research Society*, *63*, 1126–1142.

Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, M. (2012). Pre-marshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications*, *39*, 8337–8349.

Feillet, D., Parragh, S., & Tricoire, F. (2019). A local-search based heuristic for the unrestricted block relocation problem. *Computers and Operations Research*, *108*, 44–56.

Galle, V., Barnhart, C., & Jaillet, P. (2018a). Yard crane scheduling for container storage, retrieval, and relocation. *European Journal of Operational Research*, *271*, 288–316.

Galle, V., Manshadi, V., Borjian Boroujeni, S., Barnhart, C., & Jaillet, P. (2018b). The stochastic container relocation problem. *Transportation Science*, *52*, 1035–1058.

Hottung, A., Tanaka, S., & Tierney, K. (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers and Operations Research*, *113*.

Hottung, A., & Tierney, K. (2016). A biased random-key genetic algorithm for the container pre-marshalling problem. *Computers and Operations Research*, *75*, 83–102.

Jin, B., Zhu, W., & Lim, A. (2015). Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research*, *240*, 837–847.

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

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

Ku, D., & Arthanhari, T. (2016). Container relocation problem with time windows for container departure. *European Journal of Operational Research*, *252*, 1031–1039.

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.

Lee, Y., & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers and Operations Research*, *34*, 3295–3313.

Lee, Y., & Lee, Y.-J. (2010). A heuristic for retrieving containers from a yard. *Computers and Operations Research*, *37*, 1139–1147.

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.

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

de Melo da Silva, M., Toulouse, S., & Wolfler Calvo, R. (2018). A new effective unified model for solving the pre-marshalling and block relocation problems. *European Journal of Operational Research*, *271*, 40–56.

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.

Petering, M., & Hussein, M. (2013). A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *European Journal of Operational Research*, *231*, 120–130.

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*, 141–173.

Tanaka, S., & Mizuno, F. (2018). An exact algorithm for the unrestricted block relocation problem. *Computers and Operations Research*, *95*, 12–31.

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.

Tanaka, S., Tierney, K., Parreño-Torres, C., & Alvarez-Valdes, R. (2019). A branch and bound approach for large pre-marshalling problems. *European Journal of Operational Research*, *278*, 211–225.

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.

Tricoire, F., Scagnetti, J., & Beham, A. (2018). New insights on the block relocation problem. *Computers and Operations Research*, *89*, 127–139.

United Nations Conference on Trade And Development (2020). Container port throughput, annual. `https://unctadstat.unctad.org/wds/TableViewer/tableView.aspx?ReportId=13321`. Accessed: 23 March 2020.

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

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.

Zhao, W., & Goodchild, A. (2010). The impact of truck arrival information on container terminal rehandling. *Transportation Research Part E*, *46*, 327–343.

Zweers, B., Bhulai, S., & van der Mei, R. (2020). Optimizing pre-processing and relocation moves in the stochastic container relocation problem. *European Journal of Operational Research*, *283*, 954–971.

Appendix A. Proof of Lemma 1

Lemma 1. *There exists an optimal path P with the following properties:*

1. *The bays in between the leftmost and rightmost bay are visited in either increasing or decreasing order.*
2. *The endpoint of the tour is either the leftmost or rightmost visited bay.*
3. *The bay visited directly after the start bay is either the leftmost or rightmost visited bay.*

PROOF. To prove that there exists an optimal path P with these three properties we go through these properties one by one. For Properties 1 and 2, we consider an optimal path P' for which these properties do not hold and show a contradiction. For the last property we show that an optimal path P' satisfying the first two but not the last property can be changed into a path P that satisfies all three properties without changing the objective function. We denote the leftmost and rightmost visited bay by, respectively, B_l and B_r.

1. For sake of contradiction, let us consider an optimal path P' in which the first property does not hold. Without loss of generality, let us assume that bay B_l is visited before bay B_r. Let bay B_m be the first bay that is visited in path P' after bay B_n with $n > m$. On top of that, we denote the bay that is directly visited before bay B_n in path P' as B_k and the bay that is visited right after bay B_m as bay B_o. Hence, the crane travels in path P' from bay B_k to B_n to B_m to B_o and this partial route takes $t_{kn} + t_{nm} + t_{mo}$ time units. If the order of bays B_m and B_n is swapped, then the total travel time to visit these bays is $t_{km} + t_{mn} + t_{no}$. As it is given that $t_{mn} = t_{nm}$ and that both $t_{km} \leq t_{kn}$ and $t_{no} \leq t_{mo}$, we know that swapping the bays B_k and B_l in path P' reduces the travel time of that tour, which contradicts the optimality of path P'.
2. For sake of contradiction, consider an optimal path P' that satisfies the first property, but that does not end at the rightmost or leftmost visited bay. Let us denote the bay at which path P' ends by B_e. Without loss of generality, we assume that the rightmost visited bay B_r is visited later than the leftmost visited bay B_l. Furthermore, because of the first property we assume that the bays between B_l and B_r are visited in increasing order. We will show that visiting bay B_e before bay B_r reduces the cost of path P'. We distinguish between two different cases: the first case is the situation in which bay B_e is the only bay visited after bay B_r, and in the second case, multiple bays are visited after bay B_r.

   Consider the situation in which bay B_e is the only bay visited after bay B_r. Let us change the path P' into path P such that bay B_e is visited in between bay B_l and B_r. Using the first property, the optimal position in the path can be easily determined. Let us denote the bay that is visited before bay B_e in the new path by B_d and the bay that is visited after B_e in path P by B_f. Note that bay B_f might be equal to bay B_r. The increase in cost of visiting bay B_e in between bay B_e and B_f, instead of going directly from B_d to B_f equals $t_{de} + t_{ef} - t_{df}$. Moreover, after bay B_r, bay B_e does not need to be visited anymore which decreases the cost by t_{re}. Hence, the difference in cost by visiting bay B_e between bays B_d and B_f instead of after bay B_r equals

$$t_{de} + t_{ef} - t_{df} - t_{re} < t_{ef} - t_{re} < 0.$$

Here, the first inequality holds because $t_{de} < t_{df}$. The second inequality holds with equality if B_f is bay B_r. However, if B_f is not B_r, then by definition bay B_f is closer to bay B_e than B_r is to B_e and thus $t_{ef} < t_{re}$.

Now consider the case in which there are one or multiple bays visited in between bay B_r and bay B_e. Let bay B_g be the bay visited directly before bay B_e and bay B_h be the bay directly before B_g. Again, we will show that the cost of the tour decreases if bay B_e is visited between B_d and B_f. Note that now both B_f and B_h could be the same as bay B_r. The cost in path P' for going from B_d to B_f and from B_h via B_g to B_e equals

$$t_{df} + t_{hg} + t_{ge},$$

which can be rewritten into

$$\begin{aligned} t_{df} + t_{hg} + t_{ge} - t_{he} + t_{he} &= t_{df} + t_{de} + t_{ef} - t_{df} + t_{he} \\ &= t_{de} + t_{ef} + t_{he} \\ &\geq t_{de} + t_{ef} + t_{hg}. \end{aligned}$$

The last expression is exactly the cost of the tour if bay B_e is visited in between B_d and B_f instead of after B_h. Therefore, the cost of P' can be reduced, which is in contradiction with the optimality of path P'.

3. Consider an optimal path P' in which the third property does not hold, but the first two are satisfied. Again we assume without loss of generality that bay B_l is visited before B_r. Let us define the bay that is visited directly after the start bay of the crane, namely B_s, in P' as B_t and let bay B_u be the bay visited after bay B_t. Consider a path P in which bay B_t is not visited directly after bay B_s, but in the order that is suggested with the first property, between bays B_d and B_f. Again, note that bays B_u and B_d might be the same as B_l. The costs for going from B_s to B_t to B_u and from B_d to B_f in path P equals $t_{st} + t_{tu} + t_{df}$. In the equation below we see that these costs are exactly the same as $t_{dt} + t_{tf} + t_{su}$, which are the costs in path P' of visiting bay B_t between bays B_d and B_f and bay B_u directly after B_s:

$$t_{st} + t_{tu} + t_{df} = t_{st} + t_{tu} - t_{su} + t_{su} + t_{df} = t_{dt} + t_{tf} - t_{df} + t_{su} + t_{df} = t_{dt} + t_{tf} + t_{su}.$$

Hence, we can change path P' into a path P that satisfies the third property without changing the length of the path.