Noname manuscript No.
(will be inserted by the editor)

A GRASP approach for solving the Blocks Relocation Problem with Stowage Plan

Received: date / Accepted: date

Abstract In this paper, we present a method for finding high quality solutions for the Blocks Relocation Problem with Stowage Plan (BRLP). The interest for this problem comes from the fact that previous research has shown that a significant amount of savings can be achieved, if the process of specifying the loading sequence takes into account both the state of the bay and the stowage plan. In this work, we propose a two step greedy algorithm for the BRLP. In the first one, a heuristic is used to select the next container to be loaded into the vessel. In the second step, a new heuristic is used to relocate obstructing containers. The solutions acquired in this way are improved using a correction procedure. The idea of the correction procedure is to use a heuristic approach to recognize undesirable properties of a solution and remove them. Finally, the method is extended towards the GRASP metaheuristic. Our computational experiments show that the proposed approach manages to significantly outperform existing methods for the BRLP.

Keywords maritime shipping · blocks relocation problem · stowage plan · heuristics · GRASP

1 Introduction

In the last several decades, there has been an almost unprecedented increase in international trade. Close to 90% of it is carried out by the international shipping industry. The vast majority of it passes through container terminals, gigantic logistic centers, which serve as transshipment points. Their main purpose is to serve as temporary storage points, which are used for unloading containers from very large transport vessels and transferring them to smaller vessels, vehicles or trains for further distribution. This process is also performed in the opposite direction. Due to the enormous scale of operations at container terminals, an increase in efficiency can produce considerable savings in energy consumption and consequently positive environmental

Address(es) of author(s) should be given

effects. Due to the limited amount of storage space at terminals, containers are piled up at the container yard in such a manner to increase space utilization. This is done through block stacking (Kim and Hong, 2006), which directly affects the loading and unloading operations. This relation is formalized using several abstractions like the Blocks Relocation Problem (BRP), the Re-Marshalling Problem (RMP), i.e. intra-block marshalling, and the Pre-Marshalling Problem (PMP) (Steenken et al., 2004; Caserta et al., 2011a; Tanaka and Tierney, 2018).

In recent years, there has been an extensive amount of research dedicated to the BRP. This is, to a large extent, due to its close relation to the practical problem of loading containers onto vessels at container terminals. Except for the basic version of the problem, several variations have been considered. Some examples are the use of different objective functions (Lee and Lee, 2010; Zhu et al., 2010; Hussein and Petering, 2012b; Schwarze and Voß, 2015; da Silva Firmino et al., 2016) and sets of constraints (Zhu et al., 2010; Caserta et al., 2012; Petering and Hussein, 2013; Expósito-Izquierdo et al., 2014). The BRP has also been explored in a three dimensional setting (Lee and Lee, 2010; Forster and Bortfeldt, 2012). An important extension of the BRP is the on-line version in which the arrival of containers is also considered (Wan et al., 2009; Borjian et al., 2013; Tang et al., 2015). In the vast majority of the published research, the problem inputs are the initial state of the container yard and a given loading sequence. The goal of the BRP is usually to minimize the number of relocation operations (movement of containers) needed to retrieve the containers in the order of the loading sequence.

In this group of research papers, the relation between the stowage plan and the loading operations in the yard is disregarded. The stowage plan, or in other words the location of containers (or groups of containers) in the vessel, represents an essential part of container transport. Finding a good stowage plan is a complex problem as it needs to satisfy a wide range of constraints related to vessel stability and strength, transport of hazardous materials and efficient ship operations during visits to multiple ports (Pacino et al., 2012; Tierney et al., 2014). Generally, in container ports, the list of containers for discharging and loading to vessels is known in advance. Firstly, all the containers are unloaded and afterwards, based on the stowage plan, a loading sequence is determined and the loading process starts. It has been shown that the loading sequence can significantly affect the handling cost in the yard (Kim and Lee, 2015). In the paper by Ji et al. (2015), the Blocks Relocation Problem with Stowage Plan (BRLP), an extension of the BRP, is introduced. Note that in Ji et al. (2015), the problem was called the Block Relocation and Load Problem indicating the established abbreviation BRLP. In this version of the problem, the stowage plan is given as an input instead of the loading sequence. The results in this paper indicate that a significant amount of savings can be achieved if the process of specifying the loading sequence takes into account both the state of the yard bay and the stowage plan.

The goal of the BRLP is to minimize the number of movements needed to retrieve the containers. The difference between the BRLP and the BRP is that there are only constraints on the loading sequence instead of a fixed one used in the BRP. In the BRLP, the loading sequence only needs to enable the stowage plan. For example, if a container is placed below another container in the vessel, the former should be loaded before the latter, but no such constraints are necessary if they are placed apart.

In practice, the increased complexity of the BRLP comes from the fact that it is also necessary to find the optimal loading sequence in addition to finding the optimal movements of the containers inside the yard. Indeed, the BRP can be viewed as a special case of the BRLP, as will be explained in the next section. To the best of our knowledge, in the currently published research, there are still no exact algorithms for solving the BRLP. Note that existing integer programming formulations for the BRP (Wan et al., 2009; Caserta et al., 2012; Petering and Hussein, 2013; Zehendner et al., 2015) can be adapted to the new setting by changing the constraints related to when the retrieval of a container is allowed.

In this paper, we propose a greedy heuristic approach for the BRLP. Further, we analyze a new type of blockings that occur in the BRLP. Based on this analysis, several heuristics are defined. More precisely, we define specific rules for selecting the container that will be retrieved next and how to relocate the containers above it. The rules are designed to address specific properties existing in the BRLP. This type of approach has proven very successful in case of the BRP (Expósito-Izquierdo et al., 2014; Jovanovic and Voß, 2014) and the PMP (Expósito-Izquierdo et al., 2012; Jovanovic et al., 2017).

To enhance the performance of the basic greedy algorithm, a local search, more precisely a correction procedure is developed. Finally, a basic randomization is included and a GRASP algorithm (Feo and Resende, 1995) is implemented. The performed computational experiments show that the proposed approach significantly outperforms existing methods.

The paper is organized as follows. In Section 2, the problem definition and an overview of related research are given. In the next section, we give an outline of the proposed greedy algorithm. The fourth section gives a detailed analysis of different types of blockings that can occur in the BRLP. In the following section, we analyze heuristic functions for the BRLP. Further, we give details of the proposed GRASP algorithm and the correction procedure. Finally, we show the results of the performed computational experiments.

2 Definition of the BRLP

In this section, we give the definition of the BRLP as in (Ji et al., 2015). In the BRLP, it is assumed that all the containers are of the same size. The problem settings are as follows.

- The yard bay is viewed as a two dimensional stacking array with W stacks and a maximal allowed height (number of tiers) H.
- The vessel bay consists of W^v stacks. Each of them has a maximal allowed height (tier) H^v_i.
- Each container c has a designated location in the vessel bay, specified with a vessel stack $vs(c)$ and a vessel tier $vt(c)$.
- The initial configuration of the yard bay and the designated locations of containers in the vessel bay are known in advance. We will use the notation $ys(c)$/ $yt(c)$ to indicate the stack/tier in the yard bay of container c.
- Only containers from the top of a stack can be accessed.

- When a container is retrieved, it is moved from the yard bay to the vessel bay.
- The relocation operations (movement of containers within the bay) are only allowed while a target container needs to be retrieved. More precisely, only relocation of containers above the target container is allowed. In case of the BRP, this version of the problem is commonly called restricted. In practice, it means that no pre-marshalling (preparatory movement of containers in the yard bay) is allowed. This is a common method of operations at container terminals.
- Once a container is moved to the vessel bay, it cannot be moved again.
- Containers can only be placed on top of other containers or on the ground (tier 0) in both the vessel bay and the yard bay.

The objective of the BRLP is to minimize the number of relocation operations needed to move all the containers from the yard bay to their designated locations in the vessel bay. The inputs for the problem are the stowage plan and the initial state of the yard bay. An illustration is shown in Fig. 1. In practice, the BRLP can be regarded as solving several BRPs in parallel, where each vessel stack gives us a separate loading list. The BRP can also be viewed as a special case of the BRLP, in which the vessel bay has only one stack. In case of the BRP, the problem is often extended to yards having multiple bays, or, in other words, to a three dimensional setting. Each bay represents a two-dimensional array, and the yard has multiple bays which correspond to a third dimension. The BRLP can also be extended to this setting in a similar way.

A2	B3	C4	D4	E3	F2
A1	B2	C3	D3	E2	F1
	B1	C2	D2	E1	
		C1	D1		
A	**B**	**C**	**D**	**E**	**F**

E2	D3		D2	
C1	D5	D1	B1	C4
B3	A2	C3	E3	E1
A1	B2	F2	D4	C2
1	**2**	**3**	**4**	**5**

Fig. 1 Top: stowage plan. Bottom: initial state of the yard. If two blocks have the same letter, they are designated to same vessel stack. The corresponding number represents the designated tier of a container in the vessel bay.

The problem proposed in Ji et al. (2015) represents an extension of the BRP. It has been shown that the BRP is NP-hard (Caserta et al., 2012). A wide range of methods have been developed for solving this problem and its variations. Optimal solutions for the BRP have been found by solving integer programs (Wan et al., 2009; Caserta et al., 2012; Petering and Hussein, 2013; Zehendner et al., 2015). Such solutions have also been found using branch-and-bound (Kim and Hong, 2006; Tanaka and Takii, 2016; Tanaka and Mizuno, 2018) and A* (Zhu et al., 2012; Expósito-Izquierdo et al., 2014) based algorithms. Another group of methods is dedicated to finding near optimal solutions for the BRP. One part of this research focuses on developing greedy algorithms and corresponding heuristics (Zhang, 2000; Murty et al., 2005; Wu and Ting, 2010). It has been shown that the performance of such methods can be significantly improved by incorporating different types of look-ahead mechanisms (Jovanovic and Voß, 2014; Petering and Hussein, 2013; Caserta et al., 2009). Near optimal solutions for the BRP have been found using a wide range of metaheuristic methods (i.e., the beam-search (Wu and Ting, 2010; Nishi and Konishi, 2010), the corridor method (Caserta et al., 2011b) and genetic algorithms (Hussein and Petering, 2012a)).

As previously stated, different variations of the BRP have been considered. In the restricted version, only containers above the target container (the one currently being retrieved) can be relocated. There is no such constraint in the unrestricted version. Note that in case of the BRP with unique due dates, the target container is equivalent to the container with the lowest value of the due date of all the containers in the bay. This is the only container that can be retrieved based on the loading list. In case of the BRLP for each non-filled vessel stack, there is a container that can be retrieved and is potentially a target container. To avoid this ambiguity, we must select one such container and consider it the target one.

Several extensions of the BRP have been defined to better reflect the operations in the container yard. In (Forster and Bortfeldt, 2012; Lin et al., 2015; Ünlüyurt and Aydın, 2012; da Silva Firmino et al., 2016), instead of minimizing the number of relocations of containers, the goal is to minimize the crane operational time. In the work of Hussein and Petering (2013), weights are added to containers and the objective is to minimize the fuel consumption. In Zehendner et al. (2017), the BRP is considered in a setting where only incomplete information regarding the retrieval sequence is known in advance. Similar extensions of the BRLP, as in the case of the BRP, would result in more realistic representations of the real-world problem. Note that at least one optimal solution for the basic version of the BRP, where the objective is to minimize the number of relocations, is frequently the same or highly similar to the ones found for alternative objectives. An extensive analysis of the relation between optimal solutions for different variations of the BRP can be found in Schwarze and Voß (2015).

The BRP has also been extended to incorporate a wider range of operations in container yards. A direct extension is the three dimensional version, where multiple bays are considered (Ji et al., 2015; Lee and Lee, 2010). In such models, either the relocation of containers between bays is not allowed or additional costs are given to such moves. In Guerra-Olivares et al. (2015), the relocation problem employing reachstacker vehicles as container handling equipment is modeled. It considers the

full layout of the yard in determining how far a container is relocated. An important stochastic version of the BRP is the dynamic container relocation problem. It jointly considers the relocation, retrieval and stacking of incoming items (Wan et al., 2009; Tang et al., 2015). Another direction of extending the BRP is the consideration of the hinterland transport. In Ku and Arthanari (2016), such a model is developed by considering truck appointment systems.

3 Outline of the greedy algorithm

In this section, we give an outline of the greedy algorithm for the BRLP. In case of the restricted BRP, with unique due dates, the order of retrieving containers is known in advance. Due to the fact that only containers on top of stacks can be accessed, in some cases, the retrieval cannot be performed until obstructing containers are relocated. Here, we use the term *obstructing* for the containers that are above the container that is being retrieved. In the greedy algorithms for the restricted BRP, the only heuristic H_R that needs to be defined is the one for selecting a destination yard stack S for relocating an obstructing container c. As previously stated, the order in which containers are retrieved is not fully specified; instead, it only needs to satisfy some constraints in the BRLP. To be more precise, each container needs to be retrieved (loaded into the vessel) in such a sequence that all the containers below it (in the vessel bay) are already retrieved at the time of its retrieval. As a consequence, in case of the BRLP, a second heuristic H_L is defined for selecting which container will be retrieved. Pseudocode for the proposed greedy algorithm for the BRLP can be seen in Algorithm 1.

Algorithm 1 Pseudocode for the greedy algorithm for the BRLP.

```
while Bay not empty do
    Select Container a for retrieval using H_L
    while a not on top of stack do
        Select relocation stack for obstructing container b based on H_R
        Relocate b to selected stack
    end while
    Retrieve a
end while
```

According to Algorithm 1, the greedy algorithm iteratively loads containers into the vessel. At each step, a container for retrieval is selected using the heuristic function H_L. Next, all the necessary relocations are performed based on H_R. This procedure is repeated until all the containers are loaded into the vessel.

Although some heuristics H_R used for the BRP can be directly extended to the BRLP, their performance can be enhanced by focusing on specific properties of the BRLP. The second heuristic function H_L has similarities to ones developed for the PMP and can be improved in a similar way.

4 Blocking relations in the BRLP

In this section, we analyze the relations between containers in the yard bay and explain some differences between the BRP and the BRLP. The conclusions of this analysis are exploited in defining the heuristic functions for the BRLP. We first observe the relation between due dates in the BRP and designated locations of containers in the vessel bay in the BRLP. The second part is dedicated to the analyses of container relocations that increase the number of container relocations needed to retrieve containers. Further, a method for efficiently recognized such relocations is presented. The main focus is on different types of blockings that can occur in the BRLP.

4.1 Due date

In case of the BRP, each container c has a due date which is an integer value $dd(c)$. These values indicate the order in which containers are retrieved. Specifically, we can only retrieve the container with the lowest value of the due date in the yard bay. In relation, let us define the current due date for container c as $cdd(c) = dd(c) - dd(lr) - 1$, where lr is the last retrieved container. In this notation, it is evident that a container can only be retrieved when $cdd(c) = 0$.

As previously mentioned in case of the BRLP, there is no specific order in which containers are retrieved, but such a relation exists among containers inside the same vessel stack. More precisely, the containers designated to the same vessel stack are retrieved in the order of their vessel tiers. Let us note that the order in which containers are retrieved, indicated using a due date, is essential in defining several heuristics for the BRP. As previously mentioned, the proposed heuristics for the BRLP will be based on ones used for the BRP. Because of this, with the intention of having a uniform terminology, we will extend the concept of due dates to the BRLP in the following way:

$$dd_S(c) = \begin{cases} vt(c) & vs(c) = S \\ N + 1 & vs(c) \neq S \end{cases} \tag{1}$$

In Eq. (1), $dd_S(c)$ is the due date of a container c related to a vessel stack S. In case of $vs(c) = S$, the due date is equal to $vt(c)$. Otherwise, the value of the due date, related to the vessel stack S, is assumed to be equal to $N + 1$, where N is the number of containers in the initial yard bay, since it can be retrieved after all the containers designated to S. Note, the constraints related to the retrieval of such a container are independent of containers designated to vessel stack S, so it can also be retrieved before them. We also define the simplified notation $dd(c) = dd_{vs(c)}(c)$, and $cdd(c) = dd(c) - dd(lr_{vs(c)}) - 1$ where lr_s is the last container loaded to vessel stack S. Now, as in the case of the BRP, a container c can be retrieved if $cdd(c) = 0$.

4.2 Blocking and cycles

In case of solving the BRP and defining corresponding heuristics, it is essential to be able to recognize the relation between containers that result in additional relocations

when retrieving them. Let us call a container c *well-located* if there is no container d with a lower due date below it in the yard bay. Otherwise, we say a container c is not well-located. Let us use the term "container c is directly blocking container d", if $dd(c) > dd(d)$ and container d is below container c in the same yard stack. It is evident that container c will have to be relocated before d can be retrieved. In case of the BRLP, the same relation is true only if $vs(c) = vs(d)$, as seen in Fig. 2. Now, we can say a container c is well-located in the BRLP if there is no container d below it satisfying $vs(d) = vs(c)$ and $vt(d) < vt(c)$. In case such a container d exists below c, we will say c is directly blocking container d.

The difficulty in the BRLP is that there are other situations when additional relocations will be necessary. If we look at Fig. 2, it is evident that the containers $A1, A2, C1$ and $C2$ cannot be retrieved without additional relocations. In general

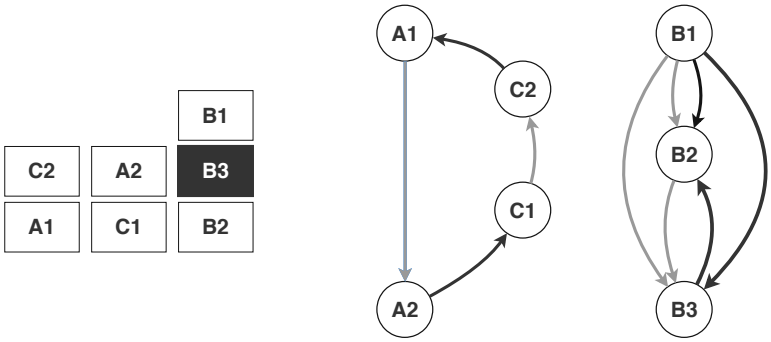

Fig. 2 An example of a precedence graph for a yard bay. The precedence corresponding to the vessel bay (relation $\xrightarrow{v}$) are grey and to the yard bay (relation $\xrightarrow{y}$) are black. The non-well-located container is colored black.

terms, an additional relocation will occur in case that for retrieving a container a, we have to previously retrieve/relocate a container b, and prior to retrieving a container b, we have to retrieve/relocate a container a. It should be pointed out that there may be some additional containers that also need to be retrieved/relocated between the retrieval of containers a and b. To recognize such situations, the concept of precedence relations and precedence graphs will be introduced.

4.2.1 Precedence graph

From the previous discussion, it is evident that there is an order in which containers can be accessed inside the yard bay and in which they can be retrieved. This order can also be observed as a precedence relation between containers. There are two types of precedence relation. The first one is based on the due dates inside of a vessel stack. For instance, container $A2$ cannot be retrieved before container $A1$ is retrieved since it cannot be placed in the designated location in the vessel bay. Let us define the

following relation

$$c \xrightarrow{v} d = \begin{cases} \top & vs(c) = vs(d) \wedge vt(c) < vt(d) \\ \bot & otherwise \end{cases} \quad (2)$$

Eq. (2) provides a formulation of the vessel precedence between containers c and d. A vessel precedence relation exists if containers c and d are designated to the same vessel stack and the designated location of container c is below that of container d. The second type of precedence is related to the positions of containers in the yard stacks, using the following relation.

$$a \xrightarrow{y} b = \begin{cases} \top & ys(a) = ys(b) \wedge yt(a) > yt(b) \\ \bot & otherwise \end{cases} \quad (3)$$

Eq. (3), states that a container a is yard precedent to container b if a is above b in some stack S in the yard bay. We can now define the general precedence relation:

$$a \rightarrow b = (a \xrightarrow{v} b) \vee (a \xrightarrow{y} b) \quad (4)$$

Using the relation $\rightarrow$, we can define a directed graph $G = (V, E)$, where each element of the node set V corresponds to a container. An edge (a, b) exists in E if $a \rightarrow b$ is true. Let us refer to this type of graph as a precedence graph. An illustration of such a graph and a corresponding yard bay configuration can be seen in Fig. 2.

4.2.2 Cycles in a precedence graph

As previously stated, an additional relocation will be necessary if for retrieving a container a, we have to previously retrieve/relocate a container b, and vice versa. Obviously, such a situation is related to a cycle in the precedence graph. It is worth mentioning that each such situation can be related to several cycles due to the way the precedence graph is constructed. To avoid this problem, we introduce the *minimal cycle*. A cycle is minimal if it does not have a proper subset of containers that can form a cycle. Examples of cycles and minimal cycles can be seen in Fig. 3. Let us prove several properties of minimal $n - cycles$ composed of exactly n containers.

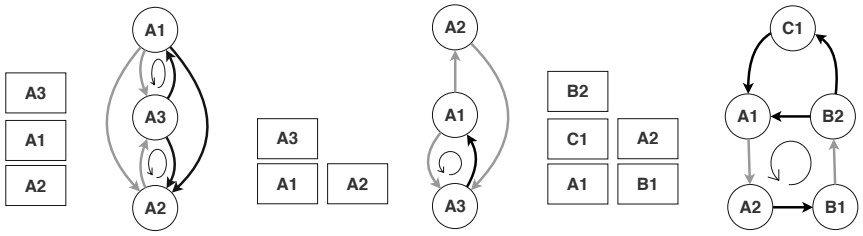

Fig. 3 Examples of cycles and minimal cycles in the precedence graph for corresponding bays. In the precedence graphs, edges corresponding to the relation $\xrightarrow{v}$ are gray and to the relation $\xrightarrow{y}$ are black. In each of the corresponding graphs a cycle exists containing all the containers which is not a minimal cycle. In each of them, the minimal cycles are indicated using the cycle symbol.

Proposition 1 *Two consecutive edges e_1 and e_2 in a minimal cycle C, cannot both correspond to the same type of precedence relation.*

Proof Let us assume that edge e_1 corresponds to vessel precedence $x \xrightarrow{v} a$ and e_2 corresponds to vessel precedence $a \xrightarrow{v} y$. It is impossible that C is a minimal 2-cycle, or in other words $x = y$, since $vt(a) < vt(x)$ and $vt(x) < vt(a)$ cannot both be true. In case $x \neq y$, from the definition of the relation $\xrightarrow{v}$, it is evident that the relation $x \xrightarrow{v} y$ will also be satisfied. Consequently, in the minimal cycle C, the two edges $x \xrightarrow{v} a$ and $a \xrightarrow{v} y$ can be substituted with edge $x \xrightarrow{v} y$, which results in a shorter cycle. The same argument holds true for yard precedence. This contradicts the minimality of C.

Proposition 2 *A minimal cycle must have $2n$ containers.*

Proof Let us first prove that if a minimal cycle C contains a container a_1, there is at least one more container $a_2 \in C$, such that $vs(a_1) = vs(a_2)$ and one of the edges $a_1 \xrightarrow{v} a_2$ or $a_2 \xrightarrow{v} a_1$ is a part of C. If there were no such container then container a_1 could only be connected to other containers in the minimal cycle using yard precedence relations which contradicts Proposition 1.

Secondly, we can prove that a minimal cycle contains at most two containers from the same vessel stack. Suppose that a minimal cycle C exists having the form $... \rightarrow a_1 \xrightarrow{v} a_2 \rightarrow ... \rightarrow a_3 \rightarrow ...$ such that $vs(a_1) = vs(a_2) = vs(a_3)$ and $a_1 \neq a_2 \neq a_3$. If $a_1 \xrightarrow{v} a_3$ is true, then $... \rightarrow a_1 \xrightarrow{v} a_3 \rightarrow ...$ is a shorter cycle than C. If $a_3 \xrightarrow{v} a_1$ is true, than the cycle $... \rightarrow a_3 \xrightarrow{v} a_1 \xrightarrow{v} a_2 \rightarrow ...$ is a shorter than C. Now, it can be trivially shown that a minimal cycle C consist of an array of pairs of containers belonging to the same vessel stack. Consequently, C has $2n$ elements. The same argument holds for yard precedence.

From this proof it is evident that the following stronger property of minimal cycles is true.

Proposition 3 *A minimal $2n$-cycle is composed of containers from n distinct vessel (yard) stacks, and there are exactly two containers in each vessel (yard) stack.*

A direct blocking corresponds to a minimal 2-cycle. If the 2-cycle contains containers a and b then the blocking container a will satisfy $a \xrightarrow{y} b$. The same concept of blocking containers can be extended to minimal $2n$-cycles. For the sake of simplicity, in the later text, we will use the term $2n$-cycle only for minimal ones. A container a in the $2n$-cycle C will be called $2n$-blocking if there is a container $b \in C$ such that $a \xrightarrow{y} b$ and the container b will be called $2n$-blocked container. It is evident from the previous discussion that all the containers in a $2n$-cycle are blocking or blocked. It should be pointed out that the total number of minimal $2n$-cycles in G is not equal to the minimal number of relocations needed to retrieve all the containers. A simple illustration is given in Fig. 4.

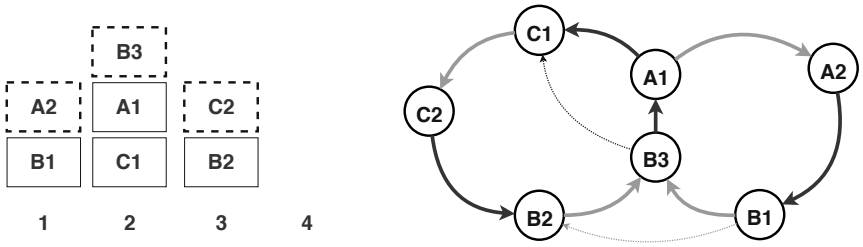

Fig. 4 An example of two 4-cycles and a corresponding precedence graph. The 4-cycles are $(A1, A2, B1, B3)$ and $(C1, C2, B2, B3)$. The 4-blocking containers are marked with a dashed line. In the precedence graph, the irrelevant edges are dotted. It is evident that by relocating container $B3$ to stack 4 both 4-cycles will be disconnected.

4.3 Relevant 4-cycles

Evaluating if a container a is well-located is trivial: we only need to check if there is a container b in the same yard stack S below a such that $dd_{vs(a)}(a) > dd_{vs(a)}(b)$. On the other hand, checking if a container a is $2n$-blocking is significantly more complex. More precisely, we need to check if the precedence graph contains a $2n$-cycle containing container a. This can be done by using the standard depth first search (DFS) approach. The problem is that it is computationally expensive. On the other hand, we can say that direct blocking (2-blocking) is more important than $2n$-blocking ($n \geq 2$) since the latter does not necessarily add relocation operations to the solution. Because of this, in the proposed method, we only check if a container is 2-blocking or 4-blocking, since this can be done without a substantial increase in computational cost. From our experience, we have seen that $2n$-cycles, where $n > 2$, are significantly less frequent. As it will be indicated below, $2n$-cycles, where $n > 2$, can frequently be avoided using the proposed correction procedure.

Let us first make a few observations regarding 4-cycles. In case a 4-cycle contains a container a that is not well-located, it will not directly result in adding a relocation to the solution. This is due to the fact that container a will have to be relocated (independent of the 4-cycle) and as a consequence the 4-cycle will be disconnected. In practice, this means that the 4-blocked/blocking container will have to be relocated before its retrieval/relocation. The situation where the 4-cycle could potentially add an additional relocation to the solution will never occur in practice.

Let us call a 4-cycle C a *relevant 4-cycle* if it only contains well-located containers. Checking if a container c is in a relevant 4-cycle can be done using a DFS to find cycles of length four in the precedence graph and checking if all the containers in it are well-located. The DFS starts from container c and has a maximal depth of five. The number of visited containers in the DFS can be significantly lowered if we exploit the fact that there are exactly two nodes in a vessel or yard stack inside the 4-cycle.

5 Heuristics

In this section, we give a detailed analysis of the two types of heuristics used in the greedy algorithm. The first one is used for selecting a stack to which an obstructing container will be relocated. The second one is used for selecting the container that will be retrieved next.

5.1 Relocation heuristics

We first present several relocation heuristics for the BRLP. For the sake of clarity, we will first state the definition of the heuristic for the BRP and later extend it to the BRLP.

The most commonly used heuristic in container ports is the lowest tier (LT) heuristic (Zhang, 2000). In the LT, a container is relocated to a yard stack with the lowest occupied tier. This heuristic has the same form for the BRP and the BRLP. The idea behind this is to avoid having stacks with many containers since this can result in a large number of obstructing containers.

One of the most effective strategies in case of the BRP is the $MinMax$ heuristic. In it, situations when relocating a container makes it well-located are treated differently, than in the case it does not. To be more precise, there is a preference to relocate a container c to a yard stack S, in which the minimal due date of a container d satisfies $dd(c) < dd(d)$. For simplicity of presentation, we define the due date for a yard stack in the following way.

$$dd(S) = \begin{cases} N+1 & S \text{ is an empty stack} \\ \min_{i \in S} dd(i) & \text{otherwise} \end{cases} \tag{5}$$

Equation (5) states that the due date of a yard stack S is equal to the minimal due date of a container in S, or $dd(S) = N + 1$ in case of an empty stack. The notation N is used for the maximal due date in the yard bay.

In the $MinMax$ heuristic, if there are several destination stacks where a container c becomes well-located, the stack S having the minimal value $dd(S)$ is selected. In case, container c must be relocated to a stack where it is not well-located, the yard stack S having the maximal value of $dd(S)$ is selected.

The $MinMax$ heuristic can be naturally extended to the BRLP. In the previous text, we have described how due dates and the concept of well-located containers can be adapted to the BRLP by taking into account vessel stacks. The only major difference in the $MinMax$ heuristics is that we take into account the vessel stack $vs(c)$ of the container c that is being relocated. Let us assume that we are relocating a container c, we can define the due date for a yard stack S using the following function:

$$dd(S, c) = \begin{cases} N+1 & S \text{ is an empty stack} \\ \min_{i \in S} dd_{vs(c)}(i) & \text{otherwise} \end{cases} \tag{6}$$

Eq. (6) gives the adaptation of the due date for stack S in case of the BRLP by extending dd with a second parameter corresponding to container c that is being relocated. In case stack S is empty, the value of $d(c, S)$ is equal to $N + 1$, where N is

the number of containers in the initial yard bay. Otherwise, it is equal to the minimal value of $dd_{vs(c)}(i)$ of all containers $i \in S$. Now the $MinMax$ heuristic for the BRLP can simply be extended from the one for the BRP by substituting the function $dd(S)$ with $dd(S, c)$.

The problem with such a direct extension of the $MinMax$ heuristic to the BRLP is that a potentially large number of yard stacks will be viewed as equivalent. To be more precise, for a container c, empty stacks and all stacks not containing any containers with designated vessel stack $vs(c)$ will be considered equally desirable. An efficient way to avoid this is to use a similar approach as in the Reshuffle Index heuristic (RI) which was originally proposed for the BRP (Murty et al., 2005). In this heuristic, there is a preference in selecting stacks with a higher value of the due date, or the closely related current due date. The logic behind this is to postpone additional relocations as much as possible, with the hope that the yard bay will come to a configuration where the container can be well-located. In case of equal values of the $MinMax$ function for the BRLP, the yard stack having the maximal value of the current due date $cdd(S)$ will be selected. Here $cdd(S)$ will be equal to the minimal value of $ddc(c)$ for $c \in S$.

As previously stated, the objective of $MinMax$ is to avoid creation of unnecessary relocations by always attempting to move a container to a yard stack S where it will be well-located. As already mentioned, in case of the BRLP, unnecessary relocations can also be created when a container is relocated to a location where it becomes 4-blocking. It is important to emphasize that such relocations do not necessarily have to increase the number of relocations needed to retrieve all the containers. One example is if the corresponding 4-blocked container is not well-located. Contrary to this, an additional move is always added to the solution if a container is placed to a location where it is not well-located.

Taking this into account, we can define a new heuristic function $MinMax4CB$. As in the case of $MinMax$, it will prefer relocating a container to a yard stack where it will be well-located. The difference is that it will differentiate between such stacks, by preferring ones were the container will not be 4-blocking. In case when it is necessary to relocate the container to a yard stack where it creates a direct blocking, it will be the same as $MinMax$. Otherwise, when a container is relocated to a yard stack where it is well-located, as in the case of $MinMax$, stacks having a small value of the due date will be preferred but this should be less important than the creation of a 4-cycle. Formally, we will select the yard stack having the minimal value of the following function:

$$dd4CB(S, c) = dds(S, c) + M \cdot Creates4Cycle(S) \tag{7}$$

In Eq. (7), the parameter M satisfies $1 \ll M$. The function $Creates4Cycle(S)$ is equal to one if the container c will create a relevant 4-cycle after relocating to yard stack S, and zero otherwise. This function can be evaluated by applying the method presented in the above section for the precedence graph corresponding to the updated yard bay. The only change is that for the stack containing the target container t, only containers below it will be considered in the precedence graph, since the rest will be relocated in the following steps. An illustration of the precedence graph for an updated yard bay can be seen in Fig. 5.

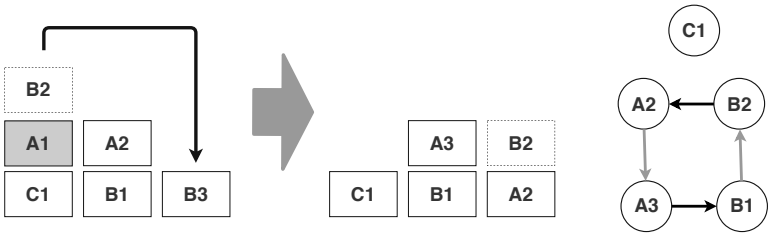

Fig. 5 Checking if a relocation of container results in a 4-blocking. The left side shows the initial yard bay from which the container $A1$ is retrieved and the obstructing container $B2$ is relocated to the third stack. The right side shows the updated bay used in the evaluation and the corresponding precedence graph.

5.2 Retrieval Heuristics

In this section, we present heuristics for determining which container will be selected for retrieval. As already stated, only a container c having $cdd(c) = 0$ can be retrieved. In practice, for each unfilled vessel stack, there will be a container satisfying this constraint. The idea of the proposed heuristic functions is to select one such container whose retrieval will require the minimal number of relocation operations.

The simplest approach is to select the container c that has the minimal number of obstructing ($MinB$) containers, using the following equation:

$$MinB = H^T_{ys(c)} - yt(c) \tag{8}$$

The logic behind $MinB$ is that when relocating a smaller number of containers there is a lower probability of creating new unnecessary relocations. In Eq. (8), H^T_S is used for the current highest occupied tier of stack S in the current yard bay.

This heuristic can be improved if we observe the properties of obstructing containers. It is preferable to relocate a container that is not well-located to one that is, since the former will have to be relocated at some step of the algorithm. On the other hand, if a container is well-located, it may become retrievable after retrieving some other container. We can define the following heuristic to take this into account:

$$MinW(c) = M \cdot AboveWL(c) + AboveNWL(c) \tag{9}$$

In Eq. (9), M is a predefined constant satisfying $1 \ll M$. The function $MinW(c)$ selects the container obstructed with the minimal number of well-located containers. In case several containers are blocked with an equal number of such containers, the one having the smallest number of non-well located containers above it is selected.

This type of heuristic can be further improved by taking into account if some of the well-located containers are 4-blocking. Let us point out once more, that it is only necessary to relocate one of the containers in a cycle in the precedence graph G to disconnect it, and as a consequence, it might not be necessary to relocate a well-located 4-blocking container. Because of this, the extended heuristic function is defined as follows.

$$MinW4CB(c) = M \cdot AboveWL^*(c) + N \cdot Above4CBlocking(c) + AboveNWL(c) \tag{10}$$

In Eq. (10), N and M are predefined constants satisfying $1 \ll N \ll M$. The heuristic states that we first prefer retrieving containers obstructed with the minimal number of well-located containers (excluding 4-blocking ones), then the minimal number of relevant 4-blocking ones and finally non-well-located ones.

5.3 GRASP

With the aim of improving the performance of the proposed algorithm, it has been extended to the GRASP metaheuristic. To achieve this task, a basic randomization has been added to the greedy algorithm and a local search, based on a correction procedure, has been developed. In our implementation, in case of both heuristics, we would randomly select one of the containers or stacks that had the minimal value of the heuristic function. It is important to note that such situations are very frequent in case of the BRLP which is not the case for the BRP.

5.3.1 Correction Procedure

In this subsection, we give details of the proposed correction procedure. In general, greedy algorithms are based on one or more heuristic functions that should guide us to good solutions. The problem is that such heuristics are often "short sighted". This can be avoided by including some look-ahead in the heuristic function. The problem with this type of approach is that the speed of the greedy algorithms is often lost due to the significant computational cost of the look-ahead function. Moreover, even look-ahead heuristics still make "mistakes". Because of this, in case of the BRP and the PMP, methods that widen the search space like beam search, A* and different tree procedures have been developed. Such methods significantly outperform the basic greedy approach, but at a high increase in the computational cost.

The goal of the proposed correction procedure is to explore a relatively small number of alternative solutions. We assume that the chosen heuristic function is "good" and only occasionally makes mistakes. The idea is that such mistakes can easily be recognized by observing the solution generated using the greedy algorithm. To be more precise, an additional heuristic will be used to decide if in some iteration of the algorithm the selected relocation is a "suspicious" one.

We will first present the criterion for recognizing potential mistakes and later a mechanism for avoiding them. Let us assume that our solution Π is a list of M moves of containers within the yard bay, or from the yard bay to the vessel bay. Each move will have the following properties: $Type$ indicating if it is a retrieval or a relocation; its index I which indicates the iteration of the move; the container r that is being retrieved or its retrieval is being made possible. Next, one has to consider the container m that is being relocated and the corresponding destination yard stack S; finally, the state of the container after relocation, or in other words if it is well-located or not.

Through extensive analysis of solutions generated using the greedy algorithm, we have observed that lower quality solutions often had the same characteristics that were the source of an increase in the number of relocations. Based on them, we have

defined the following undesirable properties of a solution, that are computationally inexpensive to recognize:

- A container c has been relocated at least M_M times. The undesirability of this property comes from the fact that there is a high probability that by initially relocating container c to one of its later locations would have been preferable. In this way, the total number of relocations in the solution can be decreased by avoiding intermediate relocations.
- A large number of obstructing containers has to be relocated when a container c, that has been previously relocated, is being retrieved. In case a solution has this property, it is highly probable that the initial savings achieved by well-locating container c have been lost in later steps of the algorithm. Informally, the relocation of container c has wasted a highly desirable location in the yard bay.
- At the time of relocation a container c has been placed to a yard stack where it is well-located, and it is relocated again before retrieval. This property of a solution indicates that the heuristic function did not give "correct" information, and as a consequence it is reasonable to try alternatives.

Using the following criteria, we can easily recognize suspicious moves, that are a source of such bad behavior:

- (CMM) M is the first relocation of a container c relocated at least M_M times.
- (CMB) M is the last relocation of a container c for which the retrieval process needed to relocate at least M_B containers.
- (CMS) M is a move at which a container was well-located (including 4-blocking) and later relocated.

The idea of the correction procedure is to remove the moves having undesirable properties from a generated solution. It is not possible to simply substitute an undesirable move in the solution with a new one since the latter moves may become invalid. Because of this, all the moves after the substituted one must be regenerated using the presented greedy algorithm. Consequently, the new solution will have different moves after the substituted one. Since the new solution is without the "suspicious" move, there is a potential that it is better than the starting one. This simple procedure can be applied to the same solution using different alternative moves and testing different undesirable ones.

When implementing this basic idea, there are several things that need to be considered. First, it is reasonable to avoid testing alternative moves that are clearly "bad" since this can significantly lower the computational cost. In case an improved solution has been found, we wish to try correcting it instead of the starting one. Because of this, the suspicious moves should be tested in the order they occur in the solution. The correction procedure based on this idea is described in the pseudocode shown in Alg. 2.

In the proposed algorithm, the main loop iteratively goes through all the moves in the best solution, starting from the first one. At each iteration, it first checks if the $CurrentMove$ satisfies any of the three suspicion criteria. In case this is true, several alternative moves are tested in the inner loop. A new partial solution is set to all the moves in the $BestSolution$ before the $CurrentMove$. A new relocation $NewMove$

Algorithm 2 Pseudocode for the correction procedure.

```
Set CurrentMove to beginning of BestSolution
while BestSolution.Length > CurrentMove.I do
    Except.Empty()
    if SuspiciousStep(CurrentMove) then
        Except.Add(CurrentMove.S)
        repeat
            CurrentSolution = BestSolution[1..CurrentMove.I − 1]

            NewMove = H_R(CurrentSolution, Except)
            CurrentSolution.Add(NewMove)
            Complete CurrentSolution using Greedy Algorithm

            Except.Add(NewMove.S)
        until NextMove.State ≠ WellLocated ∨ FoundNewBest
        Advance CurrentMove
    end if
end while
```

is selected using the relocation heuristic H_R, based on the partial solution, without considering the stacks in $Except$. $Except$ is initially set to the $CurrentSolution.S$ and later extended with tested stacks. Next, the greedy algorithm is used to complete the solution. This procedure is repeated until a new best solution is found or until the $NewMove$ state is not well-located.

With the intention of having a better evaluation of the proposed correction procedure, we give an estimate of the worst case computational complexity in relation to the basic greedy approach. For simplicity, let us assume that the computational cost of each relocation operation in the greedy algorithm is constant and equal to m, and that the total number of relocations in the solution is N. In the worst case, all N relocations in the solution are suspicious. For each suspicious move, a maximum of $W - 2$ alternative yard stacks can be selected, where the current container c can be well-located. At the i-th iteration of the main loop, the maximal number of steps that will be performed, in the reconstruction of the solution, is $N - i$. From this analysis, we can conclude that the total computational cost of the correction procedure is $m(W-2)\frac{(N-1)N}{2}$. This means that in the worst case, the computational cost of the correction procedure will be $(W-2)\frac{(N-1)}{2}$ times as large as the one of the greedy algorithm.

5.3.2 Implementation

In this subsection, we give implementation details of the proposed algorithm. For the sake of clarity, we present it the form of pseudocode in Alg. 3. In the main loop of the algorithm, new solutions are repeatedly generated until the stopping criterion is satisfied. First, the initialization steps are performed. Consider that for having an efficient implementation, it is necessary to store some calculated values and recalculate them only when necessary. This is done for the property of being well-located for containers and the precedence graph.

Algorithm 3 Pseudocode for the GRASP algorithm.

```
Initialize random generator
while Not Stopping Criteria do
    Initialize Calculations
    while Bay not empty do
        while Exists container (cdd(c) = 0) do
            Retrieve c
            Sol.Add(c, Retrive)
            Update precedence graph
        end while
        Select Container a for retrieval using Ĥ_L
        while a not on top of stack do
            Select relocation stack S for obstructing container b based on Ĥ_R
            Relocate b to S
            Sol.Add(b, Relocate, a, S, State)
            Update WellLocated information for b
            Update precedence graph
        end while
    end while
    CorrectionProcedure(Sol)
    Check if Sol is new best solution
end while
```

In the next loop, an initial solution is generated. In it, all the containers without obstructing containers above them are retrieved, if possible. At this stage, all the necessary changes are performed on the auxiliary structures. It is important to note that, although the use of function H_L would have the same effect as retrieving first the non-blocked containers, it would be more computationally expensive than just checking the value of function $cdd(c)$. After each retrieval, the current move with all the relevant properties, is added to the current partial solution Sol.

At the next step, the container a for retrieval is selected based on the randomized version of heuristic function $\hat{H}_L$. The next loop relocates all obstructing containers using the randomized function $\hat{H}_R$. As before, the auxiliary structures are updated and the performed move is added to the partial solution Sol.

After a complete solution is generated, the correction procedure is applied to it and we check if the improved solution is the new best.

6 Computational Experiments

In this section, we present the results of the performed computational experiments. Two types of evaluations have been done. The method is implemented in C# using Microsoft Visual Studio 2015. The computations have been done on a machine with Intel(R) Core(TM) i7-2630 QM CPU 2.00 Ghz, 4GB of DDR3-1333 RAM, running on Microsoft Windows 7 Home Premium 64-bit.

6.1 Comparison of GRASP to GA-ILSRS

In our first group of experiments, we compare the proposed GRASP and Greedy algorithm (with correction) (GR-C) to the $GA-ILSRS$ method from Ji et al. (2015). The comparison has been done on the same problem instances as used in Ji et al. (2015). The GRASP and GR-C algorithms use the $MinW4CB$ as the retrieval heuristic and $MinMax4CB$ as the relocation heuristic. As it will be shown in the later subsections, this was the best performing combination of heuristics. The $GA-ILSRS$ uses a genetic algorithm for finding the loading sequence. Further, it uses an improved version of the lowest tier heuristic, which takes into account if containers are being well-located to retrieve the containers based on the selected loading sequence.

The comparison is performed on the same instances as in the mentioned article for the case of a single quay crane. The container yards, in the test instances, have between 34 and 490 containers. In each of the problem instances, the yard consists of multiple bays and containers cannot be moved from one bay to another. The existence of multiple bays affects the candidate list (stacks that are being selected for relocation of an obstructing container) used with the heuristic H_R. It is important to point out that in case of 4-cycle analysis, used in H_R and H_L, the whole yard needs to be considered.

Table 1 Comparison of the performance of the proposed GRASP algorithm with GA-ILSRS

Containers	B × S × T	GA-ILSRS			GR-C			GRASP		
		Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
34	3 × 6 × 4	12	18	15.2	8	10	9.3	8	8	8.0
44	3 × 6 × 4	1	3	1.8	0	0	0.0	0	0	0.0
58	4 × 6 × 4	14	20	17.6	13	14	13.2	13	13	13.0
66	4 × 6 × 4	3	5	3.7	0	0	0.0	0	0	0.0
76	4 × 6 × 4	7	10	8.3	4	10	6.2	4	4	4.0
88	6 × 6 × 4	2	4	2.7	0	0	0.0	0	0	0.0
100	6 × 6 × 4	44	50	47.1	22	26	24.3	20	20	20.0
106	6 × 6 × 4	40	48	43.9	22	27	25.0	19	21	19.7
118	6 × 6 × 4	3	5	3.6	1	1	1.0	1	1	1.0
132	6 × 6 × 4	54	59	56.4	26	39	29.8	23	25	24.1
218	12 × 6 × 4	110	112	111.2	89	95	92	86	88	87.0
303	18 × 6 × 4	151	154	152.3	100	110	105.3	92	96	94.2
490	24 × 6 × 4	265	269	267.4	229	239	235	222	229	226.1

As in Ji et al. (2015), multiple (10) runs, using different random seeds, are executed on each of the 13 problem instances, for both the $GRASP$ and the GR-C algorithm. In the correction procedure, the number of relocations that are considered suspicious is $M_M = 2$. The number of obstructing containers that is considered suspicious is $M_B = 1$. In case of GRASP, 100 initial solutions are generated. The results of the comparison can be seen in Table 1, where the minimal, maximal and average number of relocations for each problem instance are given.

From the results in Table 1, it can be seen that the GR-C manages to significantly outperform $GA-ILSRS$ when both the average and minimal number of relocations are considered in all the tested instances. The improvement for many instances is greater than 30%. There are several reasons for this improvement. First, the performance of the greedy algorithm is highly dependent on the chosen relocation heuristic. As it will be seen in the later subsections, the lowest tier heuristic, which is used as a basis of the heuristic in GA-ILSRS, has a significantly worse performance than $MinMax4CB$. Next, as it is discussed in Ji et al. (2015) the potential number of loading sequences is very large, and the GA is used to explore them in an efficient way. For each generated loading sequence, the solution is found using a greedy algorithm, and the length of the found solution is used as the fitness function in the GA. In this way, the loading sequence and relocation operations are observed relatively independently. In case of the proposed greedy approach, the loading sequence is generated iteratively based on the current bay state, which depends on the performed relocation operations. In this way, the generation of the loading sequence and the relocation operations are closely connected, which produces a positive effect. Finally, it is well known that the performance of greedy algorithms can significantly be improved by using a local search which, contrary to GA-ILSRS, the proposed method incorporates. The repeated runs of the greedy algorithm, in the GRASP, further improve the quality of the found solutions. The improvement is most significant in the larger problem instances and is often 5-10%. This indicates that the solution space has a large number of local optimal solutions.

6.2 Comparison of different heuristics

The objective of the second group of tests is to compare different heuristics for the BRLP. To be able to do so, a wide range of random problem instances have been generated. The number of containers N in each instance is between 10 and 390. The maximal value of N is selected to be close to the number of containers inside a ship bay of the largest vessels having 18 000 TEU (twenty-foot equivalent unit), as discussed in Ji et al. (2015). In the generated problem instance, the number of vessel stacks VS is set to 3, 5, 10, 15 and 20. The vessel bay has the same structure as in Fig. 1. For each vessel bay, two yard bays have been generated having a maximal allowed tier YT of 6 or 8. The number of yard stacks YS is selected so that the total occupancy would be around 66%. For each (N, VS, YS, YT), 40 different random yard bay configurations have been generated. The generated problem instances are made available and can be downloaded from Jovanovic (2017).

We compare the lowest tier (LT), MinMax (MM) and MinMax with consideration of 4-blockings ($MM4CB$) for relocation heuristics. Each relocation heuristic has been combined with one of the heuristics for retrieval. The tested heuristics are based on the number of minimal obstructing containers including information regarding only well-located containers (MBW) and both well-located and 4-blocking containers ($MBW4CB$), as described in the Heuristics section. For each of the tested combination of heuristics, a single run has been executed for each of the 40 problem instance in one size. The average number of relocation operations can be seen

in Table 2. We did not include the results of using $MinB$, for the retrieval heuristic, since it had significantly worse results than the other two heuristics. We wish to point out that the use of the reshuffle index, as described above, is extremely important for the performance of both MM and $MM4CB$. In Table 2, we did not include execution times, since all the greedy algorithms were very fast, taking less than 20 milliseconds in the case of problem instances with 390 containers.

The first thing that can be noticed, from the results in Table 2, is that the greedy algorithm using $MM4CB + MBW4CB$ manages to outperform all the other combinations of heuristics in a vast majority of problem sizes. In only a few problem sizes, having less than 50 containers, it did not produce the best results. The use of information regarding 4-cycles, as in heuristics $MM4CB$ and $MBW4CB$, significantly improves the performance. The improvement is greater when this information is used for the relocation heuristic, compared to the retrieval one. In case of problems having more than 100 containers, this improvement is generally around 10-20% /0.1-5% in case of the relocation/retrieval heuristic. It is important to note that, $LT + MBW$ always performed worse than $MM4CB + MBW4CB$.

On the other hand, when comparing the greedy algorithm based on heuristics that do not consider 4-cycles ($MM + MBW$), this was not the case. In case of bays having more than a 100 containers, LT would get better average results than $MM + MBW$ in around 50% of the tested cases. It can be concluded that there is no significant difference between the use of heuristics LT and MM, which is contrary to the case of BRP, where the latter has a significant advantage. The main reason for this is the "stinginess" of the MM heuristic. To be more precise, the MM heuristic tries to preserve all yard stacks to which containers with a low due date (from the same vessel stack) can be well-located. This has a consequence, that containers are potentially placed to yard stacks where there is a large number of containers from other stacks, which can result in the creation of a 4-cycle. This unintentional creation of 4-cycles is a lot less frequent in case of LT, since placing a container on a yard stack with a low number of containers decreases the possibility of both direct blocking and 4-cycles. It is important to note that, if the reshuffle index is not used to break ties between stacks, in both MM and $MM4CB$, this type of issue becomes extreme and greatly reduces performance.

6.3 Evaluation of the correction procedure

In our last group of computational experiments, we evaluate the performance of the correction procedure and the corresponding GRASP algorithm. The tests are conducted on the same data sets as in the case of evaluating the heuristic functions. For each problem instance, a single run of the GRASP algorithm is performed. We consider GRASP algorithms based on heuristics $MM + MBW$ ($GRASP_N$) and $MM4CB + MBW4CB$ ($GRASP_{4CB}$). For both of them, 100 initial solutions are generated. The parameters for the correction procedure are the same as in the case of the comparison with GA-ILSRS. With the goal of having a way to evaluate the effect of the correction procedure, we also observe the results acquired using the randomized basic greedy algorithm ($MM4CB + MBW4CB$) with heuristics exploiting informa-

Table 2 Comparison of the performance of solution quality of different heuristics for the BRLP.

N	VS	YS	YT	$LT + MBW$	$MM + MBW$	$MM4CB + MBW$	$MM + MBW4CB$	$MM4CB + MBW4CB$
10	3	3	6	2.35	2.18	2.23	2.18	2.23
16	3	4	6	6.10	5.60	5.68	5.53	5.60
16	3	3	8	8.98	8.20	8.15	8.20	8.25
31	3	8	6	13.28	12.35	12.18	11.93	11.58
31	3	6	8	20.08	16.80	17.43	17.00	17.25
46	3	12	6	23.00	21.70	19.10	20.53	18.78
46	3	9	8	33.63	28.35	28.93	27.90	28.63
19	5	5	6	3.93	3.53	3.68	3.50	3.50
19	5	4	8	5.88	5.38	5.40	5.08	5.10
29	5	8	6	7.78	7.83	6.85	7.08	6.68
29	5	6	8	11.98	10.28	9.95	9.40	9.68
54	5	14	6	21.13	20.20	18.33	18.23	16.88
54	5	11	8	28.23	24.98	23.60	23.30	22.38
79	5	20	6	36.95	34.98	29.40	31.88	28.13
79	5	15	8	53.95	46.00	41.45	44.13	40.30
50	10	13	6	11.28	11.80	10.68	10.15	9.55
50	10	10	8	14.05	13.65	12.18	12.05	11.40
70	10	18	6	21.13	20.35	18.60	17.33	16.73
70	10	14	8	26.83	24.30	22.80	21.85	20.83
120	10	30	6	45.90	47.15	40.95	42.20	39.05
120	10	23	8	66.35	62.23	52.05	58.28	51.18
170	10	43	6	72.23	75.85	62.73	71.60	61.08
170	10	32	8	107.25	100.13	84.43	93.90	79.68
94	15	24	6	23.88	23.63	20.90	20.75	19.18
94	15	18	8	34.38	31.80	29.05	29.08	26.93
124	15	31	6	37.95	38.03	35.20	35.80	33.15
124	15	24	8	53.70	51.35	43.85	45.68	41.78
199	15	50	6	76.05	80.58	67.90	72.35	66.83
199	15	38	8	106.25	105.10	87.30	95.73	84.53
274	15	69	6	118.38	124.28	107.75	116.78	104.48
274	15	52	8	169.03	165.85	136.78	152.03	132.40
150	20	38	6	43.40	44.85	39.70	40.18	39.05
150	20	29	8	61.03	58.85	51.48	52.50	48.30
190	20	48	6	62.68	65.25	57.15	58.80	57.35
190	20	36	8	89.18	84.55	74.08	79.60	72.13
290	20	73	6	112.25	122.88	106.83	115.83	102.50
290	20	55	8	164.38	159.60	136.18	149.70	131.78
390	20	98	6	174.58	194.48	164.13	184.88	164.08
390	20	74	8	247.13	246.98	203.48	234.20	199.33

tion about 4-cycles. In case of the greedy algorithm, 100 separate runs have also been performed. In Table 3, the average solution qualities and execution times are presented.

From the results in this table, we can first observe that the use of the correction procedure produces a significant improvement, when compared to the $MM4CB +$

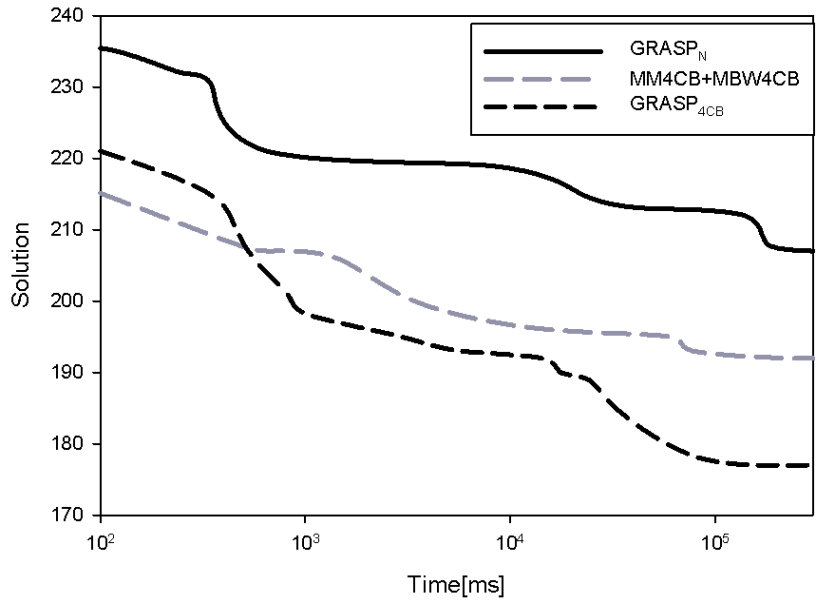

Fig. 6 Comparison of convergence speed of the proposed GRASP algorithm based on different heuristic functions and the randomized greedy algorithm.

$MBW4CB$. The improvement is relatively small (a few percent), but consistent, in case of problem instances having up to 35 containers. In case of larger problem instances, it becomes significant being generally around 10%. Although the use of the correction procedure in $GRASP_N$ produces a significant improvement when compared to the $MM + MBW$, it only manages to outperform the randomized $MM2SB + MBW2SB$ in a few of the smallest problem instances. In case of the larger problem instances, $MM4CB + MBW4CB$ manages to find solutions having between 1-15% less relocations than $GRASP_N$. This indicates that the use of the improved heuristics is essential for this type of algorithm.

The computational time of $GRASP_{4CB}$ is approximately twice as much as the one of $GRASP_N$. The computational cost of using the correction procedure largely depends on the number of relocations in the solution, as expected from the analysis of the worst case performance. The increased computational cost ranges from 2 to 40 times for the smallest to largest problem instances. This is a lower increase by several orders of magnitude, in computational cost, than the one expected from the estimate of the worst case performance. The computational cost of the greedy approach, illustrated on $MM4CB + MBW4CB$, efficiently scales. To be more precise, the increase of the computational cost of solving problems having 29 to 390 containers is around 200 times. In case of $GRASP_{4CB}$, the increase is substantially higher and is around 2500 times. Note that, although the computational cost of $GRASP_{4CB}$ is significantly higher than $MM4CB + MBW4CB$, it has a faster convergence speed. This behavior

can be observed in Fig. 6, where we illustrate the relation between solution quality and execution time for one representative problem instance having the largest number of containers.

7 Conclusion

In this paper, we have proposed a family of greedy algorithms for solving the BRLP. A wide range of heuristics have been developed and evaluated. The computational experiments have shown that by extending the heuristics standardly used for BRP and PMP to include information about 4-cycles a significant improvement can be achieved. Further, we have shown that by analyzing properties of solutions generated in this way, and avoiding undesirable behavior in them, further improvements can be achieved.

In the future we plan to explore the effect of incorporating the stowage plan into models used for housekeeping in the container yard, like the PMP. Next, we aim at defining a tight lower bound for the new problem. Another direction is to conduct an analysis of other constraints that can be added to the BRLP formulation to even better represent the real-world problem.

References

Borjian, S., Manshadi, V. H., Barnhart, C., Jaillet, P., 2013. Dynamic stochastic optimization of relocations in container terminals. In: MIT Working Paper.

Caserta, M., Schwarze, S., Voß, S., 2009. A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. In: Cotta, C., Cowling, P. (Eds.), Evolutionary Computation in Combinatorial Optimization. Vol. 5482 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 37–48.

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

Caserta, M., Schwarze, S., Voß, S., 2012. A mathematical formulation and complexity considerations for the blocks relocation problem. European Journal of Operational Research 219 (1), 96–104.

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

da Silva Firmino, A., de Abreu Silva, R. M., Times, V. C., 2016. An exact approach for the container retrieval problem to reduce crane's trajectory. In: Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on. IEEE, pp. 933–938.

Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, M., 2012. Pre-marshalling problem: Heuristic solution method and instances generator. Expert Systems with Applications 39 (9), 8337 – 8349.

Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, J. M., 2014. A domain-specific knowledge-based heuristic for the blocks relocation problem. Advanced Engineering Informatics 28 (4), 327–343.

Feo, T. A., Resende, M. G., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6 (2), 109–133.

Forster, F., Bortfeldt, A., 2012. A tree search heuristic for the container retrieval problem. In: Operations Research Proceedings 2011. Springer, pp. 257–262.

Guerra-Olivares, R., González-Ramírez, R. G., Smith, N. R., 2015. A heuristic procedure for the outbound container relocation problem during export loading operations. Mathematical Problems in Engineering 2015, article ID 201749, doi:10.1155/2015/201749.

Hussein, M., Petering, M., 2012a. Genetic algorithm-based simulation optimization of stacking algorithms for yard cranes to reduce fuel consumption at seaport container transshipment terminals. In: Evolutionary Computation (CEC), 2012 IEEE Congress on. pp. 1 –8.

Hussein, M. I., Petering, M. E., 2012b. Global retrieval heuristic and genetic algorithm in block relocation problem. In: IIE Annual Conference. Proceedings. Institute of Industrial and Systems Engineers (IISE), p. 1.

Hussein, M. I., Petering, M. E., 2013. Linear penalty relation in genetic-based algorithms in block relocation problem weights. In: IIE Annual Conference. Proceedings. Institute of Industrial Engineers-Publisher, p. 1245.

Ji, M., Guo, W., Zhu, H., Yang, Y., 2015. Optimization of loading sequence and rehandling strategy for multi-quay crane operations in container terminals. Transportation Research Part E: Logistics and Transportation Review 80, 1–19.

Jovanovic, R., 2017. Benchmark data sets for the block relocation problem with stowage plan.
URL http://mail.ipb.ac.rs/~rakaj/brlp/brlp.htm

Jovanovic, R., Voß, S., 2014. A chain heuristic for the blocks relocation problem. Computers & Industrial Engineering 75, 79–86.

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

Kim, K. H., Hong, G.-P., 2006. A heuristic rule for relocating blocks. Computers & Operations Research 33 (4), 940 – 954.

Kim, K. H., Lee, H., 2015. Container terminal operation: Current trends and future challenges. Springer, Cham, pp. 43–73, Handbook of Ocean Container Transport Logistics: Making Global Supply Chains Effective.

Ku, D., Arthanari, T. S., 2016. Container relocation problem with time windows for container departure. European Journal of Operational Research 252 (3), 1031–1039.

Lee, Y., Lee, Y.-J., 2010. A heuristic for retrieving containers from a yard. Computers & Operations Research 37 (6), 1139–1147.

Lin, D.-Y., Lee, Y.-J., Lee, Y., 2015. The container retrieval problem with respect to relocation. Transportation Research Part C: Emerging Technologies 52, 132–143.

Murty, K., Liu, J., Tseng, M. M., E. Leung, K-K. Lai, W., Chiu, 2005. Hong Kong international terminals gains elastic capacity using a data-intensive decision support

system. Interfaces 35 (1), 61 – 75.
Nishi, T., Konishi, M., 2010. An optimisation model and its effective beam search heuristics for floor-storage warehousing systems. International Journal of Production Research 48 (7), 1947–1966.
Pacino, D., Delgado, A., Jensen, R. M., Bebbington, T., 2012. An accurate model for seaworthy container vessel stowage planning with ballast tanks. Springer Berlin, pp. 17–32, International Conference Computational Logistics, Lecture Notes in Computer Science, vol 7555.
Petering, M. E., Hussein, M. I., 2013. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. European Journal of Operational Research 231 (1), 120–130.
Schwarze, S., Voß, S., 2015. Analysis of alternative objectives for the blocks relocation problem,Working Paper, Institute of Information Systems, University of Hamburg.
Steenken, D., Voß, S., Stahlbock, R., 2004. Container terminal operation and operations research - a classification and literature review. OR Spectrum, 26 (1), 3–49.
Tanaka, S., Takii, K., 2016. A faster branch-and-bound algorithm for the block relocation problem. IEEE Transactions on Automation Science and Engineering 13 (1), 181–190.
Tanaka, S., Tierney, K., 2018. Solving real-world sized container pre-marshalling problem with an iterative deepening branch-and-bound algorithm. European Journal of Operational Research 264 (1), 165-180.
Tanaka, S., Mizuno, F., 2018. An exact algorithm for the unrestricted block relocation problem. Computers & Operations Research 95, 12-31.
Tang, L., Jiang, W., Liu, J., Dong, Y., 2015. Research into container reshuffling and stacking problems in container terminal yards. IIE Transactions 47 (7), 751–766.
Tierney, K., Pacino, D., Jensen, R. M., 2014. On the complexity of container stowage planning problems. Discrete Applied Mathematics 169, 225 – 230.
Ünlüyurt, T., Aydın, C., 2012. Improved rehandling strategies for the container retrieval process. Journal of Advanced Transportation 46 (4), 378–393.
Wan, Y.-w., Liu, J., Tsai, P.-C., 2009. The assignment of storage locations to containers for a container stack. Naval Research Logistics (NRL) 56 (8), 699–713.
Wu, K.-C., Ting, C.-J., 2010. A beam search algorithm for minimizing reshuffle operations at container yards. In: Proceedings of the 2010 International Conference on Logistics and Maritime Systems, Busan, Korea, September 15-17, 2010.
Zehendner, 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 (2), 415–422.
Zehendner, E., Feillet, D., Jaillet, P., 2017. An algorithm with performance guarantee for the online container relocation problem. European Journal of Operational Research 259 (1), 48 – 62.
Zhang, C., 2000. Resource planning in container storage yard, Ph.D. thesis, Department of Industrial Engineering, The Hong Kong University of Science and Technology.
Zhu, M., Fan, X., He, Q., 2010. A heuristic approach for transportation planning optimization in container yard. In: Industrial Engineering and Engineering Man-

agement (IEEM), 2010 IEEE International Conference on. IEEE, pp. 1766–1770.
Zhu, W., Qin, H., Lim, A., Zhang, H., 2012. Iterative deepening A* algorithms for the container relocation problem. IEEE Transactions on Automation Science and Engineering 9 (4), 710–722.

Table 3 Comparison of the computational cost and solution quality of the proposed GRASP algorithm based on different heuristic functions and the randomized greedy algorithm.

N	VS	YS	YT	Solution			Time[s]		
				$MM4CB + MBW4CB$	$GRASP_N$	$GRASP_{4CB}$	$MM4CB + MBW4CB$	$GRASP_N$	$GRASP_{4CB}$
10	3	3	6	2.18	2.15	2.15	<0.01	<0.01	0.01
16	3	4	6	5.35	5.13	5.08	<0.01	0.01	0.01
16	3	3	8	8.05	7.60	7.60	<0.01	0.01	0.02
31	3	8	6	10.95	10.58	10.45	0.01	0.03	0.04
31	3	6	8	16.30	14.75	14.68	0.01	0.04	0.08
46	3	12	6	17.83	17.38	16.85	0.02	0.07	0.09
46	3	9	8	27.25	23.25	23.63	0.02	0.10	0.21
19	5	5	6	3.35	3.33	3.28	0.01	0.01	0.01
19	5	4	8	4.90	4.90	4.85	0.01	0.01	0.02
29	5	8	6	6.23	6.43	6.18	0.01	0.02	0.03
29	5	6	8	8.98	8.48	8.45	0.01	0.03	0.04
54	5	14	6	15.88	15.70	14.68	0.03	0.08	0.11
54	5	11	8	20.73	19.78	19.15	0.03	0.11	0.18
79	5	20	6	26.18	27.15	24.68	0.05	0.24	0.30
79	5	15	8	37.75	36.45	33.83	0.06	0.35	0.56
50	10	13	6	8.08	8.35	7.83	0.03	0.05	0.07
50	10	10	8	10.23	10.23	9.65	0.03	0.06	0.10
70	10	18	6	14.15	14.53	13.40	0.05	0.12	0.19
70	10	14	8	18.43	18.53	16.85	0.05	0.16	0.26
120	10	30	6	33.48	35.13	30.53	0.14	0.56	0.91
120	10	23	8	45.58	47.78	40.85	0.15	0.88	1.49
170	10	43	6	52.80	57.98	48.20	0.27	1.67	2.51
170	10	32	8	70.78	79.28	64.68	0.29	2.70	4.07
94	15	24	6	15.50	16.43	14.68	0.08	0.18	0.34
94	15	18	8	22.88	22.83	20.53	0.09	0.27	0.52
124	15	31	6	27.00	27.55	24.63	0.15	0.43	0.84
124	15	24	8	36.18	37.25	32.40	0.16	0.69	1.29
199	15	50	6	57.38	60.40	51.50	0.39	2.19	4.51
199	15	38	8	73.95	79.78	66.45	0.42	3.33	6.59
274	15	69	6	91.25	97.00	82.18	0.76	6.62	13.03
274	15	52	8	118.50	132.43	106.40	0.83	10.70	19.87
150	20	38	6	31.28	31.73	27.78	0.22	0.63	1.35
150	20	29	8	40.70	42.40	36.65	0.24	1.09	2.17
190	20	48	6	46.05	47.65	41.70	0.36	1.46	3.10
190	20	36	8	61.70	64.30	55.38	0.39	2.34	5.31
290	20	73	6	90.00	92.48	81.40	0.87	6.82	14.90
290	20	55	8	117.78	126.40	105.03	0.95	11.12	25.05
390	20	98	6	143.83	152.98	130.50	1.74	22.86	48.44
390	20	74	8	178.10	199.68	161.88	1.82	33.44	75.90