An Efficient Ant Colony Optimization Algorithm for the Blocks Relocation Problem

Raka Jovanovic

Qatar Environment and Energy Research Institute (QEERI), Hamad bin Khalifa University, PO Box 5825, Doha, Qatar

Milan Tuba

Graduate School of Computer Science, John Naisbitt University, Bulevar umetnosti 29, Belgrade, Serbia

Stefan Voß[1]

Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
and Escuela de Ingenieria Industrial, Pontificia Universidad Católica de Valparaíso, Chile

Abstract

In this paper we present an ant colony optimization (ACO) algorithm for the Blocks Relocation Problem (BRP). The method is applied to both versions of the problem most commonly considered in literature, i.e., the restricted (rBRP) and the unrestricted (uBRP) BRP with distinct due dates. In case of the uBRP a new heuristic is proposed and incorporated in a standard greedy algorithm. The performance of the basic greedy approach is enhanced by extending it to the ACO metaheuristic. In it, a novel approach for defining the pheromone matrix is proposed. More precisely, it only stores a small amount of information instead of the complete bay state. Further, we show that the proposed ACO method can easily be adapted for solving the BRP in which the objective function is related to the crane operation time. Our computational results show that the proposed approach manages to outperform existing methods for the BRP.

Keywords: heuristics, maritime shipping, blocks relocation problem, stowage plan, heuristics, ant colony optimization

Email address: `stefan.voss@uni-hamburg.de` (Stefan Voß)

1. Introduction

In the recent decades an unprecedented growth of international trade has occurred. The vast majority of it is carried out by the international shipping industry through container terminals. The main function of these terminals is to serve as transshipment and 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, and in the opposite direction. Due to the enormous amount of goods passing through container terminals, an increase in efficiency can imply considerable financial savings and a decrease in energy consumption and consequently positive environmental effects.

One of the main issues at container terminals is the limited space used for storage. As a consequence, containers are piled up at the container yard in such a way to increase the space utilization, more precisely by using block stacking (Kim and Hong, 2006). Inside the port the majority of the energy consumption is for stacking operations (68 %) and horizontal transport of containers by, e.g., tractors (30 %) (Wilmsmeier et al., 2014). The loading and unloading operations are directly related to the stacking and indirectly to the horizontal container transport. The speed of these operations is also essential for the time a vessel needs to spend docked, and is a standard measure of efficiency of a port. Because of this, a significant amount of research has been dedicated to the problem of minimizing the number of relocations of containers inside the port. For this practical problem several different models have been developed like the Blocks Relocation Problem (BRP), the Re-Marshalling Problem (RMP), i.e. intra-block marshalling, and the Pre-Marshalling Problem (PMP) (Caserta et al., 2011a; Steenken et al., 2004; Lehnfeld and Knust, 2014).

Recently, a noteworthy research effort has been dedicated to the BRP. The initial version of the BRP has been adapted in several ways to better represent the real-world problem. This relates to different objective functions (Lee and Lee, 2010; Zhu et al., 2010; Hussein and Petering, 2012b; da Silva Firmino et al., 2016; Schwarze and Voß, 2015), sets of constraints (Caserta et al., 2012;

Petering and Hussein, 2013; Expósito-Izquierdo et al., 2014; Zhu et al., 2010), three dimensional yards (Lee and Lee, 2010), an on-line version (Wan et al., 2009; Tang et al., 2015; Borjian et al., 2013), existence of uncertainties (Borjian et al., 2013; Zehendner et al., 2017; Ku and Arthanari, 2016), etc. Although being more versatile, the basic concept for solving most of these models is taken from methods developed for the original BRP.

Examples of exact methods using integer programming include Wan et al. (2009); Caserta et al. (2012); Petering and Hussein (2013); Zehendner et al. (2015). Other methods for finding optimal solutions are based on branch-and-bound (Kim and Hong, 2006; Tanaka and Takii, 2016; Zhu et al., 2012) and A* (Zhu et al., 2012; Expósito-Izquierdo et al., 2014) algorithms. Due to the NP-hardness of the problem (Caserta et al., 2012) such methods have a limit in the size of the problem instances that can be solved in time limits deemed practical. Because of this, a wide range of methods has been developed for finding near optimal solutions for the BRP and its variations.

In case of the BRP, and the closely related PMP, the use of different greedy algorithms has proven to be successful (Zhang, 2000; Murty et al., 2005; Wu and Ting, 2012; Ünlüyurt and Aydın, 2012; Jovanovic et al., 2017; Expósito-Izquierdo et al., 2012). The performance of these types of approaches has been improved by incorporating different look-ahead mechanisms (Jovanovic and Voß, 2014; Petering and Hussein, 2013; Caserta et al., 2009; Jin et al., 2015). Alternative heuristics include the use of a tree search (Forster and Bortfeldt, 2012) and a domain-specific knowledge-based algorithm (Expósito-Izquierdo et al., 2014). Moreover, a wide range of metaheuristics has been applied to the BRP, including beam-search (Wu and Ting, 2012; Nishi and Konishi, 2010), the corridor method (Caserta et al., 2011b), genetic algorithms (Hussein and Petering, 2012a), etc.

The ant colony optimization (ACO) (Dorigo and Gambardella, 1997) metaheuristic has been successfully applied for optimizing a wide range of port operations. Some examples are the optimization of the container load sequencing (Lee et al., 2005), the container stacking problem (Ndiaye et al., 2014), loading of individual containers (Zhang and Du, 2011), berth allocation (Sun et al., 2010), etc. In case of problems related to loading/unloading operations of containers, ACO has only been applied to the PMP (Tus et al., 2015). One of the main reasons for the lack of research in this direction is the difficulty of defining the pheromone matrix as the commonly used heuristic functions are related to different states of the bay whose number is enormous. In Tus et al. (2015), in case of the PMP, this has been resolved by the dynamic allocation of the pheromone matrix. One drawback of this approach is the loss of simplicity of implementation of the ACO, which is one of its main advantages. Note that in many cases the simplicity of implementation of ACO algorithms is also reflected in the reduced computational complexity; one such example is Sreeja and Sankar (2015).

In this paper we focus on developing an ACO method that avoids this problem. To make this possible we use an alternative formulation of the candidate list that is used in the related greedy algorithms. Further, a new direct heuristic for the BRP is defined which is suitable for use in the ACO transition rule. Finally, a novel approach to defining the pheromone matrix is presented. The main idea is to have the matrix in form of a multidimensional array, having a small number of dimensions, that only stores a small but important amount of information about the bay state. We show that ACO using this type of pheromone matrix can easily be extended to the BRP in which the objective function is related to the crane operation. Computational experiments show that the proposed ACO method manages to outperform other advanced methods for solving the BRP in both the quality of found solutions and the computational cost.

The paper is organized as follows. In Section 2, we give the definition of the BRP. In the next two sections, we present the greedy algorithms for the restricted and unrestricted version of the BRP. In Section 5 we give details of the proposed ACO algorithm. In the next section we show how the proposed ACO method can be extended to the BRP where the objective function is related to crane operation time. The following section provides results of our computational experiments and Section 8 concludes the paper.

2. Definition of the BRP

We present the BRP using the standard formulation. In it all containers are of the same size and the problem setting is as follows.

- The yard bay will be viewed as a two dimensional stacking array with W stacks and a maximal allowed height (number of tiers) H. $Stacks$ will be used for the set of all stacks in the yard bay.
- There is a total of N containers in the bay with each container c having a designated unique due date having a value from 1 to N.
- The initial configuration of the yard bay is known in advance. We use the notation $s(c)/t(c)$ to indicate the stack/tier of container c in the yard bay.
- Only containers from the top of a stack can be accessed (retrieved or relocated).
- When a container is retrieved, it is removed from the bay.
- Only the container c with the lowest due date can be retrieved.
- A container can only be relocated on top of other containers or on the ground (tier 0) of the yard bay.

The objective of the BRP is to minimize the number of relocations needed to retrieve all the containers from the yard bay. Alternatively, we consider the minimization of the crane operation time as an objective. In this paper we consider two versions of the problem: the restricted BRP (rBRP) and the unrestricted BRP (uBRP). The uBRP corresponds to the previously given formulation. The rBRP includes the following additional constraint.

- (RES) The relocation operations (movement of containers within the bay) are only allowed for containers which are above the target container t (t has the minimal value of the due date) currently being retrieved, which means no look-ahead or pre-marshalling. Note that throughout the paper

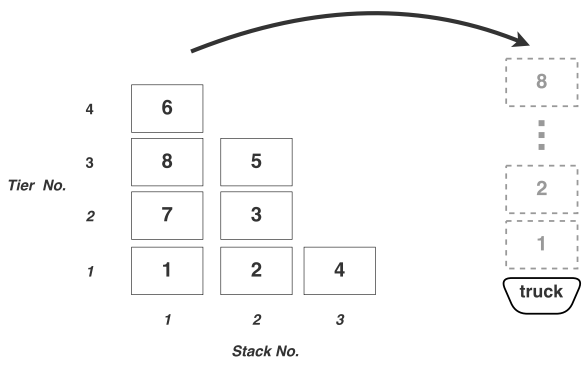

Figure 1: An illustration of the problem setting of the BRP.

we shall be using the notation t to indicate that a container is a target container.

An illustration of the BRP can be seen in Fig. 1.

3. Outline of the Greedy algorithm for the rBRP

In this section we give an outline of the greedy algorithm for the rBRP with unique due dates. In it the order of retrieving containers is known in advance. With the intention of having a simpler notation, as we are solving the BRP with unique due dates, c will be used for the due date and the container itself. We use the term *target container* for the container that is currently being retrieved (i.e. the one with the lowest due date). As 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 containers that are above the target container. In greedy algorithms for the rBRP the only heuristic h that needs to be defined is used for selecting a relocation stack S for an obstructing container c.

The basic greedy algorithm for solving the rBRP can be seen in Algorithm 1. The algorithm iteratively retrieves containers one by one. This is done by first relocating each of the obstructing containers, from the top of the stack

Algorithm 1 Pseudocode for the greedy algorithm for rBRP

```
while Bay not empty do
    Select target container $t$ having the minimal due date
    while $t$ not on top of stack do
        Select relocation stack for obstructing container $c$ based on $h$
        Relocate $c$ to selected stack
    end while
    Retrieve $t$
end while
```

containing the target container, to the stack selected using the heuristic function h. This procedure is repeated until all the containers are retrieved.

The most commonly used heuristic in container ports is the lowest tier (LT) heuristic (Zhang, 2000). In LT a container is simply relocated to a stack with the lowest tier. The idea behind this heuristic is to avoid having stacks with many containers since this can result in a large number of obstructing ones that need to be relocated later.

One of the most effective strategies for the rBRP is the $MinMax$ heuristic. In case of the BRP, a container c is called *well-located* if there is no container d with a lower due date below it. Let us use the term "container d is directly blocking container c" if $c < d$ and c is below d in a yard stack. It is evident that d will have to be relocated before retrieving c. In relation, we define the function $WellLocated(c)$ for a container c, which is equal to $true$ if c is well-located (for the current state of the bay) and $false$ otherwise.

In the $MinMax$ heuristic which was introduced by Caserta et al. (2011b), situations when a container's relocation results in it becoming well-located or not are treated differently. Firstly, it is considered most desirable to well-locate a container if this is possible. If there are multiple choices where such a relocation can be done, we choose the stack in which the minimal value of the due date of a container is the lowest. In this way we avoid wasting slots where other containers with later due dates can be well-located. In case a container must be relocated to a stack where it is not well-located, the idea is to postpone further relocation of that container as much as possible.

Formally, we prefer to relocate a container c to a stack S in which the minimal due date of a container d satisfies $c < d$, i.e., where it is well-located. In case of several such stacks, it is preferred to select one that has the lowest value of the minimal due date of a container. The idea is to preserve slots which are more valuable in the sense that more containers, having later due dates, can be well-located. In case c must be relocated to a stack where it is not well-located, the stack S having the maximal value of the minimal due date of a container is selected. The reasoning is to postpone the future relocation of a container as much as possible in the hope that a slot where it can be well-located will become available.

Let us define the heuristic function $MinMax$ in a convenient way for later use in the ACO. H_S is the highest or maximally occupied (i.e., containing a container) tier of stack S. Let us note that the symbol $\wedge$ is used for the logical "and" operation. With that we can define the set of candidate stacks for relocation, R_c (all non-full stacks without c):

$$R_c = \{S \mid (S \in Stacks) \wedge (H_S < H) \wedge (S \neq s(c))\} \tag{1}$$

As previously stated, the $MinMax$ heuristic depends on the minimal value of the due date of a container in the stack. So, let us define a function $dd(S)$ which is equal to the minimal value of the due date of all containers in stack S. In case of an empty stack S, we define this value as $dd(S) = N + 1$ as any container can be well-located if it is placed on the ground (tier 0):

$$dd(S) = \begin{cases} \min_{c \in S} c & H_S > 0 \\ N + 1 & H_S = 0 \end{cases} \tag{2}$$

Next, let us define a function $dif(c, d)$ that gives us the desirability, corresponding to our heuristic, of placing container c above container d as follows:

$$dif(c, d) = \begin{cases} d - c & d > c \\ 2N + 1 - d & d < c \end{cases} \tag{3}$$

In Eq. (3) the value of $dif(c, d)$ ranges from 1 to $2N$. Note that $dif(c, d) \leq N$ indicates that a container c can be well-located above container d and the value of $dif(c, d)$ tells us how many slots are lost. If $dif(c, d) > N$ then c will not be well-located if placed above d and the higher the value of d the lower the value of $dif(c, d)$ will be. We can extend the function dif to stacks as

$$dif(c, S) = dif(c, dd(S)) \tag{4}$$

In Eq. (4) $dif(c, S)$ uses the minimal due date of a container in stack S as the value of a container in dif. Using dif, we can select the relocation stack such that container c will be relocated to the stack having the minimal value of $dif(c, S)$ out of all candidate stacks in R_c:

$$MinMax(c) = \underset{S \in R_c}{\arg\min}\ dif(c, S) \tag{5}$$

Note that for any state of the bay, a non-empty stack can uniquely be defined by its container with the minimal due date. Using this function $dd(S)$, empty stacks are seen differently from non-empty ones, but they cannot be differentiated. Note that such a differentiation is not necessary for this heuristic function. We exploit this fact to give an alternative formulation of function $MinMax(c)$. We first define the set

$$M_c = \{d \mid (d = dd(S)) \wedge (S \in R_c)\} \tag{6}$$

representing the set of containers with minimal due dates for all the potential stacks. Now the method for selecting a relocation stack can be defined as:

$$MinMax(c) = s(\underset{d \in M_c}{\arg\min}\ dif(c, d)) \tag{7}$$

4. Greedy algorithm for the uBRP

Contrary to the rBRP, in case of the uBRP we do not know which container will be relocated at each step. To be more precise, in case of rBRP the container on top of the stack containing the target container must be relocated/retrieved, while in case of uBRP any container on top of a stack can be relocated/retrieved next. Next we provide details of the greedy algorithm for the uBRP. The basic idea of the algorithm will be the same as in Alg. 1, with the difference of a higher number of candidate relocations. It can be understood as having two parts, the first is defining the candidate list of potential relocations and the second one is the heuristic used to select one of them. Two algorithms are defined. In the first only one relocation of non-well located containers will be considered and in the second one no such restriction will exist.

4.1. Basic greedy algorithm

In this subsection we define the greedy algorithm in which only relocation of non-well-containers will be allowed. In this algorithm we use the same heuristic as in case of the rBRP, but a different list of candidate relocations which will be defined subsequently. Let us define $Tops$ as the set of all containers that are on the top of a stack in the bay, and $top(S)$ as the container at the top of stack S. A relocation is generally defined as an ordered pair $\alpha = (S, D)$ where S is the source stack and D is the destination stack. In such a relocation we know that $c = top(S)$ must be relocated, so the relocation can alternatively be defined as an ordered pair $\alpha = (c, D)$. Now the set of all potential relocations is

$$\tilde{C} = \bigcup_{c \in Tops} (\{c\} \times R_c) \tag{8}$$

Let us make several observations about the set of candidates for relocation, $\tilde{C}$. First the relocation of the container $c = top(s(t))$, which is on top of the stack containing the target container t, is always desirable since as soon as the target container is not obstructed it can be retrieved. Generally, the relocation of a well-located container should be avoided since it adds a relocation that is not necessary for retrieving all the containers. Relocation of non-well-located containers which are not above the target container is only reasonable if they are well-located after the relocation. The reason for this is that such relocations are not necessary for retrieving the target container and might be postponed.

Using these observations we can define an improved restricted list of candidates. Let us start with the following definitions:

$$
\begin{aligned}
W_c &= \{S \mid S \in R_c \wedge dd(S) > c\} && (9)\\
T_n &= \{d \mid d \in Tops \wedge \neg WellLocated(d) \wedge s(d) \neq s(t)\} && (10)\\
T &= \{top(s(t))\} \times R_{top(s(t))} && (11)\\
O_r &= \bigcup_{c \in T_n} (\{c\} \times W_c) && (12)\\
C_r &= T \cup O_r && (13)
\end{aligned}
$$

W_c is the set of all stacks to which container c will be well-located after relocation. The set T_n contains all top containers that are not well-located and not above the target container. Set T corresponds to all potential relocations for the rBRP. O_r represents the set of all relocations of non-well-located containers not above the target container to stacks where they are well-located. Finally, C_r gives the restricted set of candidate relocations for the uBRP.

In case of the uBRP the same heuristic function dif will be used as for the rBRP. The only difference in the greedy algorithm is in the candidate list from which the relocation is selected. With $\alpha = (c, S)$ and $dif(\alpha) = dif(c, S)$ defining the extension of function dif to candidate relocations, the selection function is as follows (the best relocation among all potential candidates):

$$MinMax(Bay) \quad = \quad \arg\min_{\alpha \in C} dif(\alpha) \qquad (14)$$

Let us make a few observations regarding this heuristic function. If there is no chance of well-locating any of the top containers which are not above the target container it is equivalent to the selection function used in the rBRP. Next, it always performs a relocation that well-locates a container if such a relocation exists. Out of all such moves, it performs one leading to the loss of the lowest number of slots for well-locating containers, in the same way as in the rBRP.

4.2. Extended greedy algorithm

In this subsection we present an extended version of the greedy algorithm presented in the previous subsection. To be more precise, in the new greedy algorithm the relocation of well-located containers will be allowed in some specific

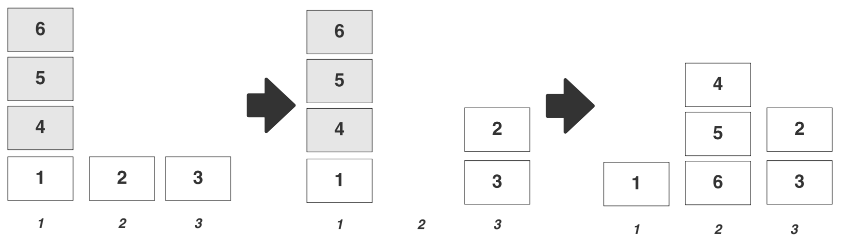

Figure 2: An illustration of a bay state for which a significant decrease in the number of relocations needed to retrieve all the containers from the bay can be achieved by relocating a well-located container. Grey containers indicate non-well-located containers. If container 2 is relocated all the obstructing containers can be well-located. Otherwise, this is not possible.

cases. We first define a list of candidate relocations and later the corresponding heuristic function. Let us start with a few observations.

A big disadvantage of the greedy algorithm based on the selection function given in Eq. (14) is that it never relocates well-located containers. Observing Fig. 2, we can see that in some cases such (look ahead) moves are highly beneficial. However, this type of move automatically adds a relocation to the solution and all the potential benefits that are immediately acquired, in the sense of making it possible to well-locate some containers, might occur anyways in later steps of the algorithm. If we relocate a well-located container c, we know that $dd(s(c))$ will increase and it might happen that some containers can now be well-located in this stack. The idea is to allow such moves only if they are highly beneficial. We consider such moves "beneficial enough" if the container $a \in T_n$ with the largest due date can be well-located in this stack. Note that there are many other potentially less strict criteria. From our experience a relocation of a well-located container is rarely beneficial, so the idea of these criteria is to only perform them if it is expected that they are highly helpful.

Because of this the case when a non-well-located container can be well-located will be treated separately from the case when this is not possible. Using this group of observations we can specify a new set of candidates for the next relocation in case it is not possible to well-locate a non-well-located container.

In the following text, we will use the notation $dd(S, h)$ for the value of the minimal due date in a stack for the lowest h tiers. Let us start with several definitions.

$$imp(c) = dd(s(c), t(c) - 1) - dd(s(c)) \tag{15}$$

$$max_n = \max_{c \in T_n} c \tag{16}$$

$$\hat{T}_w = Tops \setminus (T_n \cup \{t\}) \tag{17}$$

$$T_w = \{c \mid c \in \hat{T}_w \wedge dd(s(c), t(c) - 1) > max_n\} \tag{18}$$

$$O_w = \bigcup_{c \in T_w} (\{c\} \times R_c) \tag{19}$$

$$C_w = T \cup O_w \tag{20}$$

In Eq. (15) $imp(c)$ is defined for a container c and gives the increase of the due date $dd(S)$, where $S = s(c)$, after the top container is relocated. Eqs. (16), (17), (18) are used to define the set of top well-located containers whose relocation makes it possible to well-locate the container with the latest due date among those that are non-well-located and on the top of a stack. In (16), max_n is the maximal due date of all the stack top containers. $\hat{T}_w$ represents the set well-located containers on the top of a stack not including the target container t. T_w is the set of well-located containers for which relocation is considered based on the previously mentioned constraints. Eq. (19) defines the set O_w of all candidate relocations for well-located stack tops. Finally, C_w represents the complete set of candidate relocations which also contains relocations of the container obstructing the target container.

The next step is to define a new heuristic function that can evaluate the desirability of relocating well-located containers and ones obstructing the target container. The idea is that we wish to get a large number of new slots if a relocation of a well-located container is performed. Such a heuristic should have a preference for relocating well-located containers that stay well-located after relocation if additional slots are gained. It should also give us some method to compare the desirability of relocations of well-located and non-well-located containers. Note that the only relocation of a non-well-located container being considered is that of the container above the target container.

A heuristic function satisfying these constraints can be defined:

$$dif_e(c,S) = \begin{cases} N + dif(c,S) - imp(c) & WellLocated(c) \\ dif(c,S) & otherwise \end{cases} \tag{21}$$

In case of relocating the container obstructing the target one, the value of the heuristic function will be $dif(c,S)$. In case c is well-located we need to extend the logic of function dif to the new setting. Its value will be a sum of N (since an additional relocation is added to the solutions) and the value of $dif(c,S)$. This is reduced by the number of new slots gained by removing the top container from stack S. Note that dif can be seen as a restriction of dif_e to non-well-located containers. Eq. (21) is given such that it can be easily used in ACO.

Finally, we can define the corrected selection function, which considers both well-located and non-well-located containers, as

$$NR = \bigcup_{c \in T_n \cup \{s(top(t))\}} W_c \tag{22}$$

$$\hat{C} = \begin{cases} C_w & NR = \emptyset \\ C_r & NR \neq \emptyset \end{cases} \tag{23}$$

$$MinMax_c(Bay) = \arg\min_{\alpha \in \hat{C}} dif_e(\alpha) \tag{24}$$

In Eq. 22, we define NR as the set of all non-well-located containers that can be well-located. The set $\hat{C}$, defined in Eq. 23, is used to specify the candidate set of relocations depending on the fact if it is possible to well-located a non-well-located container or not. In Eq. (24), $MinMax_c(Bay)$ can be understood as a correction of the $MinMax$ selection function, that can also select relocations of well-located containers if the mentioned criteria are satisfied. This correction is achieved by using the extended heuristic function dif_e and the appropriate candidate set.

5. Application of ACO to the BRP

Next we present an ACO approach for solving the BRP, based on the greedy algorithms from the previous sections. In many contexts, ACO is best understood as an “intelligent” randomization of a greedy algorithm. The “smart” behavior of ACO comes from experience gained by previously generated solutions, which is stored in a pheromone matrix. The idea is to have a colony of n artificial ants. Each ant generates a solution by expanding a partial one through iterations as in the greedy algorithm. The difference comes from the use of a probabilistic transition rule, instead of the heuristic function, to decide how to expand the partial solution. As previously stated, the pheromone matrix stores the experience gathered by the artificial ants. This is achieved through the application of global and local update rules. The former are applied after all n ants in the colony have generated a solution and the goal is to reinforce the selection of elements inside the best found solution or in some variations of good solutions. The local update is performed after an ant has applied the transition rule, with the objective of diversifying the exploration of the solution space by avoiding the selection of the same elements of the solution by all the ants. In the following subsections the definition of the pheromone matrix, the transition rule, the local and global update rules, and implementation details for the proposed ACO algorithm for the BRP are given.

5.1. Pheromone Matrix

When defining the pheromone matrix for the BRP, the initial idea is to connect it to the state of the bay (see Tus et al. (2015) in case of the PMP). This type of matrix has several drawbacks due to the enormous number of potential states of the bay. In case of Tus et al. (2015), this problem is avoided by dynamically allocating only pheromone values for the encountered bay states. Although this resolves the issue of memory usage, new ones occur like the extra computational cost of accessing values in the pheromone matrix. The other drawback of such an approach is the increased complexity of the implementation. It can be argued that one of greatest appeals of ACO is the simplicity in extending a greedy algorithm to this metaheuristic. In the general case the pheromone matrix τ is a multidimensional array which is simply updated based on the elements that have been used to expand the partial solution.

In the proposed approach we attempt to maintain this simplicity while maximizing the amount of information that the pheromone matrix contains. Let us analyse how such a matrix in case of the BRP can be defined. Let us first note that the pheromone matrix in ACO is generally closely related to the heuristic function itself. This function, in case of both rBRP and uBRP, is used to evaluate which potential candidate of the form (c, S), where c is a container and S is a stack, is most desirable for expanding the solution. So it is natural that the pheromone matrix τ has elements of the form τ_{cS}. From the previous discussion we have seen that for a specific bay state a stack S is uniquely defined by the container $dd(S)$. On the other hand, we can say that $dd(S)$ provides us with additional information about the state of stack S. Because of this instead of the pheromone matrix having the form τ_{cS} we use one whose elements have the form τ_{cd}, where c is the container being relocated and $d = dd(S)$.

In case of the BRP it is common that some containers are moved multiple times. The problem of using a pheromone matrix with the form τ_{cd} is that it considers all moves the same. On the other hand, during the execution of the greedy algorithm, that is used as a base for ACO, tracking how many times a container is relocated is trivial. Because of this we extend the pheromone matrix to τ_{cdn} where the additional index n indicates how many times a container has been moved. The information contained in the pheromone matrix can be significantly increased by including the information about the current target container t. This is due to the fact that it is reasonable to expect that the state of the bay is significantly different after the retrieval of several containers.

Information contained in the pheromone matrix with the form τ_{cdnt} makes it possible to fully reconstruct a solution. The fact that such a pheromone matrix does not distinguish between all the different states of the bay, can even be beneficial in the learning process of the colony. To be more precise, several states of the bay correspond to the same value τ_{cdnt} but in many cases, from a practical point of view, they are the same for the problem being solved. By using the proposed approach, experience is gained jointly for all of them.

Let us summarize the definition of the pheromone matrix τ_{cdnt}:

- t is the current target container, c is the container being relocated. $t, c = 1..N$
- d is used to identify the destination stack using its container with the minimal due date. This definition has to be extended to differentiate between different empty stacks. So for non-empty stacks $d = dd^*(S) = dd(S)$, and for empty stacks $d = dd^*(S) = N + i(S)$, where $i(S)$ is the index of the stack. $d = 1..(N + W)$
- n corresponds to the number of times the container c has been moved. $n = 0..MaxMoves$

5.2. Transition Rule

In ACO it is convenient to have a larger value of the heuristic function for more desirable elements, so we define the following functions:

$$\alpha = (c, S) \tag{25}$$

$$f(c, d) = \frac{1}{1 + dif_e(c, d)} \tag{26}$$

$$f(c, S) = f(c, dd(S)) \tag{27}$$

$$f(\alpha) = f(c, S) \tag{28}$$

Let us note again that $dif_e(c, d)$ is equal to $dif(c, d)$ in case of a non-well-located container c. In Eqs. (27), (28) $f(c, S)$ $(f(\alpha))$ give us the extension of the function f to stacks (candidates) where the value of the minimal due date in the stack is used as the value of the container d.

In the formulation of the transition rule we assume that the number of times a container c has been previously relocated is known. The notation m_c is used to give this number. Further, at each step of the algorithm the target container t is also known. As previously stated, in an ACO algorithm the selection of the next expansion of the partial solution is done based on the heuristic function and experience stored in the pheromone matrix. Let us define the function $g(\alpha)$ which contains this information as the product of the heuristic function f and the corresponding pheromone value:

$$d = dd^*(S) \tag{29}$$

$$g(\alpha) = f(\alpha)\tau_{cdm_ct} \tag{30}$$

Using $g(\alpha)$ we can define the transition of ACO in the standard way:

$$select = \begin{cases} \arg\max\limits_{\alpha \in C} g(\alpha) & q < q_0 \\ prob & q \geq q_0 \end{cases} \tag{31}$$

In Eq. (31) *select* gives the selected relocation, where parameter q_0 is used to define the exploitation/exploration rate. Connected to it, $q \in (0,1)$ is a random variable which specifies whether the next relocation is to be selected deterministically or non-deterministically. In case of the former ($q < q_0$), we simply select the relocation α with the maximal value of $g(\alpha)$. Otherwise ($q \geq q_0$), the probability distribution for selecting a relocation is given as follows.

$$prob(\alpha) = \frac{g(\alpha)}{\sum_{\delta \in C} g(\delta)} \tag{32}$$

Eq. (32) states that the probability of selecting α is proportional to $g(\alpha)$ and inversely proportional to the sum of all $g(\delta)$ for all the candidate relocations δ. Let us note that the transition rule in case of rBRP, and uBRP using heuristics defined in Eqs. (14), (24) will only differ in the candidate list. More precisely, in case of rBRP C is equal to T from Eq. (11). In case the uBRP is solved using the heuristic function that allows the relocation of well-located containers, the candidate list C is equal to $\hat{C}$ for Eq. (23). Otherwise, if such relocations are not allowed, the candidate list will be the set C_r from Eq. (13).

5.3. Global and Local Update Rules

The next component of the ACO method that needs to be defined is the local and global update rule. To achieve this we first define a measure of quality for a solution $\mathcal{S}$ using the following equation:

$$val(\mathcal{S}) = \frac{1}{|\mathcal{S}| - LB + 1} \tag{33}$$

In Eq. (33) LB is a lower bound for the solution (number of relocations) of the BRP from the initial state of the bay. In case of the rBRP we use the lower bound proposed by Zhu et al. (2012). In case of uBRP we use the total number of containers that are not well-located in the bay. The quality of the solution is inversely proportional to the difference between the number of relocations in the solution $|\mathcal{S}|$ and the lower bound. The addition of one is used to avoid division by zero.

Before specifying the global and local update rules, the used structure of a solution for the BRP will be pointed out. In the standard formulation the solution consists of an array of pairs source/destination stack (S, D). As previously stated the pheromone matrix depends on a 4-tuple (c, d, m_c, t), which directly defines a source/destination pair based on the state of the bay. Because of this we assume that the complete/partial solution $\mathcal{S}$ is an array of such 4-tuples.

The goal of the global update rule is to intensify the exploration around high quality solutions. The proposed ACO algorithm is based on the ant colony system (ACS) (Dorigo and Gambardella, 1997), in which only the best found solution deposits pheromone after each iteration of the colony. This update is formally defined using the following equations:

$$\Delta\tau = val(\mathcal{S}_{best}) \tag{34}$$

$$\tau_{cdm_ct} = (1-p)\tau_{cdm_ct} + p\Delta\tau \quad , \forall(c, d, m_c, t) \in \mathcal{S}_{best} \tag{35}$$

In Eq. (35) $\mathcal{S}_{best}$ is used to note the current best found solution. $\Delta\tau$ is used to specify the quality of the solution $\mathcal{S}_{best}$ based on function val. Parameter $p \in (0, 1)$ is used to specify the influence of the global update rule. Note that Eq. (35) only effects the values of pheromone τ_{cdm_ct} for $(cdm_ct) \in \mathcal{S}_{best}$.

As previously mentioned the local update rule is applied after individual ants perform the transition rule. In our implementation the local update rule is applied after an ant i has generated a solution $\mathcal{S}_i$ using

$$\tau_{cdm_ct} = \varphi\tau_{cdm_ct} \, , \quad \forall(c, d, m_c, t) \in \mathcal{S}_i \tag{36}$$

where $\varphi \in (0, 1)$ is used to specify the influence of the local update rule.

5.4. Implementation

Next we present the implementation details of the ACO method for the BRP. The first step is choosing the initialization value for the pheromone matrix.

$$\tau_0 = \frac{1}{W} val(\mathcal{S}_g) \tag{37}$$

The initialization value τ_0 is calculated based on the quality of the solution $\mathcal{S}_g$ found using the greedy algorithm based on the corresponding heuristic function. There are several ACO variants, out of which, in the majority of cases, the MAX-MIN ant system (MMAS) (Stützle and Hoos, 2000) achieves the best performance. Some comparisons of ACO variations can be seen in Tuba and Jovanovic (2009) and Shishvan and Sattarvand (2015). It is an extension of the ACS in which limits are given to the minimal and maximal value of the pheromone. In the case of our ACO implementation for the BRP we only use a limit for the minimal value of the pheromone τ_{min}:

$$\tau_{min} = \frac{1}{W^2} val(\mathcal{S}_{best}) \tag{38}$$

With the goal of having a more comprehensive understanding of the proposed method, it is presented in the form of a pseudo-code given in Alg. 2. The ACO algorithm has the same form for both rBRP and uBRP with the difference being which equation is used for selecting the list of relocation candidates, as discussed in Subsection 5.2. In the method the first step is finding a solution using the appropriate greedy algorithm. Next, a lower bound LB is calculated based on the initial state of the bay. Using these two values the pheromone matrix τ is initialized. The main loop performs iterations of the ant colony until a stopping criterion is reached. At each iteration the n ants generate a solution one by one. This is done first by clearing the solution $\mathcal{S}$, resetting the container relocation counter M and other auxiliary structures. Next, containers are retrieved one by one. At each step a target container is selected and relocations are performed until it is at the top of the stack and retrieved. This is done by first calculating the candidate list using one of the Eqs. (11), (13) and (23), depending on which version of the problem is being solved. The best relocation α is selected using

Algorithm 2 Pseudo-code for the ACO algorithm for the BRP.

```
Generate solution $\mathcal{S}_g$ using the greedy algorithm
Calculate $LB(Bay)$
Initialize the pheromone matrix $\tau$ based on $\mathcal{S}_g$
while (Not Stopping Criteria) do
    for $n$ ants do
        Clear Solution $\mathcal{S}$
        Reset array for tracking the number of relocations $M$
        Reset auxiliary structures
        while Bay not empty do
            Select Container target $t$ having the minimal due date
            while $t$ not on top of stack do
                Calculate candidate list $C$
                Select best relocation $\alpha \in C$ based on transition rule
                $\mathcal{S}.Add(\alpha.c, dd^*(\alpha.S), M[\alpha.c], t)$
                Apply relocation $\alpha$ to $Bay$
                Update auxiliary structures
                if $|\mathcal{S}| + LB(Bay) \geq |S_{best}|$ then
                    $valid = false$
                    break double while        ▷ Stop generating current solution
                end if
            end while
        end while
        Apply local update rule for $S$
        Check if $\mathcal{S}$ is valid new best solution
    end for
    if $MaxConst$ iterations without improvement then
        Reinitialize pheromone matrix
    end if
    Apply global update rule for $S_{best}$
end while
```

the heuristic function. Next, the partial solution is expanded by the 4-tuple $(\alpha.c, dd^*(\alpha.d), M[\alpha.c], t)$. Finally, the relocation α is applied to the bay; the array M and other auxiliary structures are updated.

The next step in the inner loop is to check, based on the lower bound for the current bay state, if the partial solution can potentially be used to generate a solution better than the current best $\mathcal{S}_{best}$. If this is not possible, the generation of a solution is aborted. After a solution is generated or the generation is aborted the local update rule is applied for the solution $\mathcal{S}$, even if it is partial. It is important to note that the early termination of solution generation has two purposes. The first, less important, is the decrease in computational cost. The second, more important, is not wasting the information stored in the pheromone matrix for generating unnecessary low quality solutions. For each ant we check if it has generated a new best solution. After all the ants have generated their solutions, the global update rule is applied. Finally, at each iteration of the colony we check if stagnation of the algorithms has occurred and if so, the pheromone matrix is reinitialized.

Note that due to the pheromone matrix having four dimensions the proposed method can have a high memory cost. The size of the pheromone matrix is approximately N^3 times the maximal allowed number of relocations. In case of problem instances having up to a 100 containers the ACO algorithm uses around 50 megabytes of RAM. We wish to point out that the vast majority of elements in the pheromone matrix are not used. As a consequence the memory usage can be significantly decreased if dynamic allocation of the pheromone matrix is implemented. Though, for solving the standardly used problem instances this was not necessary.

6. Application of ACO to alternative objectives

In this section we present a simple approach for applying the proposed ACO method to the BRP with alternative objective functions. To be more precise, we focus on the application of the proposed approach to the rBRP in case where the objective is to minimize the working time of the crane. The main incentive for this extension is that previous research has shown that there is a close relation between the solutions of the rBRP in case the objective function is the number of relocations or the crane working time (Schwarze and Voß, 2015). The idea is to simply substitute the new objective function in the generation of the pheromone matrix but use the same heuristic function as in the basic problem.

We use two types of objectives in case of crane working times. In the first, there is a time cost for moving the crane between stacks t_s and a constant time effort t_{pp} for the pick-up/place-down of a container. In the new setting it is considered that all containers, in the retrieval, are moved to stack 0. Now we can define, as in Schwarze and Voß (2015), the following cost function:

$$\alpha = (c, S) \tag{39}$$

$$cost_f(\alpha) = \begin{cases} 2t_s s(c) + t_{pp} & \textit{Retrieval} \\ 2t_s|s(c) - S)| + t_{pp} & \textit{Relocation} \end{cases} \tag{40}$$

Eq. (40) gives the cost of operations in the rBRP. When the operation is retrieving container c, we consider the cost to be equal to the sum of the pick-up/place-down time and the time for the crane to move from the “retrieval” stack 0 to the current stack $s(c)$ of the container being retrieved, and back. In case of a relocation operation, since we are solving the rBRP, the crane will go from the current stack of container c to the destination stack S and back. Note that this is an approximation to the total movement of the crane, standardly used in literature, and not exact. To be more precise, the described movement is not exact in case the target container is changed or the last container is retrieved.

In the second objective the time of the pick-up/place-down is not fixed but dependent on the number of tiers the container crosses. In this setting we will assume there is a constant time t_r needed for a container to cross one tier. In Schwarze and Voß (2015), it is assumed that the container goes from its current tier to a maximal one h_{max}, then moves to the destination stack S to its new top tier H^*_S. At retrieval, it is assumed that the “retrieval” stack has a height

h_{out}. For such a setting the cost function has the following form:

$$cost_v(c) = \begin{cases} 2t_s s(c) + t_r(2h_{max} - t(c) - h_{out}) & \textit{Retrieval} \\ 2t_s|s(c) - S| + t_r(2h_{max} - t(c) - H^*_S) & \textit{Relocation} \end{cases} \tag{41}$$

Now, the objective function can be defined for solution $\mathcal{S}$ as

$$O_f(\mathcal{S}) = sum_{\alpha\in\mathcal{S}} cost_f(\alpha) \tag{42}$$

$$O_v(\mathcal{S}) = sum_{\alpha\in\mathcal{S}} cost_v(\alpha) \tag{43}$$

Eqs. (42),(43) simply state that the objective is equal to the sum of costs of all operations in the solution $\mathcal{S}$. Note that in the new formulation the solution also contains the relocation operations.

To be able to define an efficient algorithm it is also important to define a lower bound for the new objectives. They will be used in defining the function for evaluating the quality of a solution similar to Eq. (33). In case of a fixed pick-up/place-down time we use the following lower bound:

$$LB_f = \sum_{c\in Bay} (t_{pp} + 2s(c)t_s) + \sum_{c\in NW} (t_{pp} + t_s) \tag{44}$$

The value of LB_f for O_f has two parts. The first, corresponding to the first term in Eq. (44), is equal to the time of retrieving all the containers from their current location. The second is related to the additional cost resulting from necessary relocations. Again we use the number of non-well-located containers as the number of necessary relocations. For each such relocation, the additional cost is equal to the time needed to move the crane one stack, and perform one pick-up/place-down operation. In Eq. (44), the notation NW is used for the set of all non-well-located containers in the bay.

In case of O_v we can define a similar lower bound LB_v:

$$LB_v = \sum_{c\in Bay} ((2h_{max} - t(c) - h_{out})t_r + 2s(c)t_s) + \sum_{c\in NW} (2t_r + t_s) \tag{45}$$

The first term of Eq. (45) is related to the cost of retrieving all the containers from their current locations. The second term refers to the minimal additional cost of relocating the container. It is equal to the time needed to move the crane one stack, and to move a container over two tiers.

Finally, the proposed ACO method can be extended to the rBRP with the objective function based on the crane working time. Let us define the new quality functions val

$$val_f(\mathcal{S}) = \frac{1}{O_f(\mathcal{S}) - LB_f + 1} \tag{46}$$

$$val_v(\mathcal{S}) = \frac{1}{O_v(\mathcal{S}) - LB_v + 1} \tag{47}$$

The values of functions val_f, val_v will be used in the definition of the global update rule and the initial value of the pheromone trail. Note that LB_f, LB_v, O_f and O_v will also be used in checking if a current partial solution can potentially generate a new best solution in Alg. 2.

7. Computational Experiments

In this section we present the results of our computational experiments used to evaluate the performance of the proposed ACO. All the algorithms have been implemented in C# using Microsoft Visual Studio 2015. The calculations 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. The evaluation for both rBRP and uBRP uses the standardly applied data sets initially presented in Caserta et al. (2009). In it, each problem instance has an initial bay ($T \times S$), having T tiers and S stacks, with a total of T times S containers. The used data set contains a wide range of problem sizes ranging from 3×3 to 10×10. For each bay size there are 40 different randomly generated problem instances.

In all the computational experiments we use the standard values of the ACO parameters (Dorigo and Gambardella, 1997). The parameters for specifying the influence of the global and local update rules have the values $p = 0.1$ and $\varphi = 0.9$; $q_0 = 0.9$ is used for the exploitation/exploration rate in the transition rule. The colony has $n = 10$ artificial ants. The stopping criterion in case of

Table 1: Comparison of the average solution quality for different methods in case $H_{max} = T + 2$. The following notation is used for competing methods: the corridor method (CM) (Caserta et al., 2011b; Caserta, 2017), the proposed ACO method and optimal solutions acquired using A* (OPT)(Expósito-Izquierdo et al., 2014).

	CM	ACO		OPT
T X S		sol	time[s]	
3 X 3	5.00	5.00	<0.01	5.00
3 X 4	6.18	6.18	<0.01	6.18
3 X 5	7.02	7.02	<0.01	7.02
3 X 6	8.40	8.40	<0.01	8.40
3 X 7	9.28	9.28	<0.01	9.28
3 X 8	10.65	10.65	<0.01	10.65
4 X 4	10.20	10.20	<0.01	10.20
4 X 5	12.95	12.95	<0.01	12.95
4 X 6	14.02	14.02	<0.01	14.00
4 X 7	16.12	16.12	<0.01	16.12
5 X 4	15.42	15.42	<0.01	15.28
5 X 5	18.88	18.95	<0.01	18.65
5 X 6	22.18	22.15	0.01	21.95
5 X 7	24.25	24.33	0.02	22.08
5 X 8	27.75	27.73	0.01	-
5 X 9	30.58	30.50	0.02	-
5 X 10	33.40	33.40	0.03	-
6 X 6	31.05	31.05	0.06	-
6 X 10	46.10	45.93	0.09	-
10 X 6	79.95	79.50	0.63	-
10 X 10	115.60	113.45	1.70	-

the rBRP (uBRP) is 5000 (1000) iterations of the colony. The criterion for stagnation of the algorithm is that $MaxConst = 100$ iterations of the colony have been executed without any improvement of the best solution. In defining the pheromone matrix the maximal number of relocations of a container is $MaxMoves = 10$.

In the following subsections we first evaluate the performance of the proposed ACO method for the rBRP and later for the proposed greedy and ACO algorithms for the uBRP and for alternative objectives.

7.1. Evaluation of ACO for rBRP

In the first group of our computational experiments we analyze the performance of the proposed method for rBRP. The ACO method has been evaluated in the case when the maximal allowed tier is limited to $H_{max} = T + 2$. The comparison has been done to the corridor method (CM) proposed in Caserta et al. (2011b) and with the optimal solutions acquired using the A* algorithm (Expósito-Izquierdo et al., 2014). In case of the CM, the original code has been

Table 2: Comparison of the average solution quality for LA-N and the proposed greedy algorithm with the constraint that well-located containers cannot be relocated (Gre-N) and in case there is no such restriction (Gre-C). The values for LA-N have been taken from Petering and Hussein (2013).

T X S	LA-N	Gre-N	Gre-C
3 X 3	5.10	5.10	5.10
3 X 4	6.30	6.25	6.20
3 X 5	7.00	6.92	6.92
3 X 6	8.40	8.45	8.35
3 X 7	9.20	9.22	9.22
3 X 8	10.60	10.70	10.50
4 X 4	10.40	10.22	10.30
4 X 5	13.00	12.92	12.68
4 X 6	14.00	14.05	13.98
4 X 7	16.40	16.20	15.80
5 X 4	15.80	15.72	15.58
5 X 5	19.70	19.32	19.12
5 X 6	22.60	22.18	21.95
5 X 7	24.80	24.18	23.50
5 X 8	27.80	27.52	27.00
5 X 9	30.70	30.18	29.78
5 X 10	33.50	32.62	31.92
6 X 6	32.60	31.82	31.12
6 X 10	46.80	45.30	44.52
10 X 6	85.00	85.18	82.75
10 X 10	119.50	113.82	111.22

updated and run on a modern day machine. The code and the results can be found at Caserta (2017). As in the original paper on CM, a time limit of 60 seconds for each problem instance has been used for finding solutions. It is important to point out that the results are better than the ones presented in Caserta et al. (2011b). The main reason for this is due to the nature of the method for which longer execution times or the same execution times on a faster computer make it possible to explore a greater area of the solution space. A detailed explanation can be found in Caserta (2017). The summarized results of the performed computational experiments can be seen in Table 1. The results in this table represent the average values for 40 problem instances of the same size. From these results, it is evident that both methods have a very good performance. ACO manages to achieve better average solution quality in case of 6 problem sizes, while being worse in only 2 cases. We wish to point out that the improvement of the ACO method, compared to the CM, is most significant in case of the largest problem instances.

7.2. Comparison of simple greedy algorithms for uBRP

In the second group of tests we evaluate the performance of the proposed greedy algorithm for the uBRP by comparing it to the LA-N method from Petering and Hussein (2013) on the same group of data sets. We have chosen to compare to this method due to the fact that it is also a direct greedy heuristic. As in the proposed approach, LA-N does not include any type of tree search or backtracking. Further, it only defines a list of potential candidate relocations and chooses one based on a simple heuristic. Due to the ease of implementation of such algorithms, they are often used in the development of more complex methods or for acquiring upper bounds. The maximal allowed tier H_{max} was the same as in Petering and Hussein (2013) and had the value $H_{max} = 2T - 1$. Once more we compare the average quality of solutions for each test size. In this comparison we do not include computational times since both types of greedy algorithms are very fast. The summarized results are given in Table 2.

In this table we show the average quality of solutions for the two proposed greedy algorithms, one with the constraint that well-located containers cannot be relocated (Gre-N) and the second one where there is no such restriction (Gre-C). In case of LA-N we have used the best results presented in Petering and Hussein (2013). The method Gre-C had the best performance. In only one of the 21 tested sizes it had a worse average solution quality than LA-N, and in this case it was only slightly worse, 9.22 to 9.20. On the other hand, it managed to get better results for 19 problem sizes. The improvement was generally significant, frequently being around $3 - 5\%$. Gre-N in general performed better than LN-S but the improvement was less significant, and it had a worse performance for several sizes. This indicated that an effective greedy method for uBRP must consider the relocation of well-located containers.

7.3. Comparison of advanced approaches for uBRP

In our next group of tests we compare the performance of ACO with state-of-the-art methods for finding high quality solutions for the uBRP. To be more precise, we compare our method to the domain-specific knowledge-based heuristic

Table 3: Comparison of the average computational cost and solution quality for different methods in case $H_{max} = \infty$. For the ACO algorithm with the constraint that well-located containers cannot be relocated we use the notation ACO-N and ACO-C in case there is no such restriction. The computations have been performed on different machines.

T X S	Time[s]				Avg Solution					
	FB	EXP	ACO-N	ACO-C	Gre-C	FB	EXP	ACO-N	ACO-C	OPT
3 X 3	<0.1	0.01	<0.01	<0.01	5.00	4.95	4.95	4.95	4.95	4.95
3 X 4	<0.1	0.01	<0.01	<0.01	6.15	6.05	6.02	6.10	6.02	6.02
3 X 5	<0.1	0.01	<0.01	<0.01	6.88	6.85	6.88	6.88	6.85	6.85
3 X 6	<0.1	0.02	<0.01	<0.01	8.28	8.28	8.28	8.35	8.28	8.28
3 X 7	<0.1	0.03	<0.01	<0.01	9.15	9.20	9.10	9.18	9.15	9.10
3 X 8	<0.1	0.04	<0.01	<0.01	10.35	10.45	10.38	10.58	10.33	10.30
4 X 4	<0.1	0.02	<0.01	<0.01	10.20	9.90	9.70	9.90	9.75	9.65
4 X 5	<0.1	0.04	<0.01	<0.01	12.63	12.63	12.30	12.63	12.35	12.22
4 X 6	<0.1	0.08	<0.01	<0.01	13.80	13.70	13.38	13.60	13.35	13.22
4 X 7	<0.1	0.12	<0.01	<0.01	15.75	15.78	15.60	15.78	15.40	13.60
5 X 4	<0.1	0.07	0.01	<0.01	15.63	15.00	14.68	15.08	14.78	14.42
5 X 5	<0.1	0.18	0.02	0.02	19.13	18.63	18.02	18.30	17.78	16.84
5 X 6	<0.1	0.29	0.01	0.02	21.95	21.83	21.08	21.48	21.15	16.00
5 X 7	<0.1	0.44	0.01	<0.01	23.38	23.58	23.28	23.53	22.80	18.00
5 X 8	<0.1	0.81	0.05	0.04	26.90	26.73	26.65	26.78	26.18	-
5 X 9	0.1	1.10	0.03	0.02	29.65	29.45	29.40	29.40	28.85	-
5 X 10	0.2	1.63	<0.01	0.01	31.88	31.85	31.70	32.13	31.45	-
6 X 6	0.4	0.73	0.08	0.10	30.95	29.68	28.98	30.13	28.88	-
6 X 10	2.3	3.34	0.06	0.19	44.28	43.60	42.45	43.88	42.50	-
10 X 6	45.5	35.32	0.69	0.91	82.65	75.65	76.02	75.78	72.63	-
10 X 10	60.0	57.62	1.30	2.35	110.30	116.90	104.02	109.13	103.58	-

algorithm (EXP) presented in Expósito-Izquierdo et al. (2014), and the heuristic tree search procedure (FB) presented in Forster and Bortfeldt (2012). The comparison has been performed on the same group of data sets. The methods have been compared for $H_{max} = T + 2, \infty$. As in the previous subsections, we compare the average quality of solutions per problem size, and average computational time. The results of this comparison can be seen in Tables 3, 4. In these tables we also include the values of optimal solutions acquired using an A* algorithm from Expósito-Izquierdo et al. (2014). In these tables we include results for Gre-C and the ACO algorithm based on the greedy algorithm Gre-N

Table 4: Comparison of the average computational cost and solution quality for different methods in case $H_{max} = T + 2$. For the ACO algorithm with the constraint that well-located containers cannot be relocated we use the notation ACO-N and ACO-C in case there is no such restriction. The computations have been performed on different machines.

T X S	Time[s]				Avg Solution					
	FB	EXP	ACO-N	ACO-C	Gre-C	FB	EXP	ACO-N	ACO-C	OPT
3 X 3	<0.1	<0.01	<0.01	<0.01	5.10	4.98	4.98	4.98	$\underline{4.98}$	4.98
3 X 4	<0.1	0.01	<0.01	<0.01	6.20	6.05	$\underline{6.02}$	6.10	$\underline{6.02}$	6.02
3 X 5	<0.1	0.01	<0.01	<0.01	6.93	$\underline{6.85}$	6.88	6.88	$\underline{6.85}$	6.85
3 X 6	<0.1	0.02	<0.01	<0.01	8.35	$\underline{8.28}$	$\underline{8.28}$	8.35	$\underline{8.28}$	8.28
3 X 7	<0.1	0.03	<0.01	<0.01	9.23	9.23	$\underline{9.15}$	9.18	$\underline{9.15}$	9.10
3 X 8	<0.1	0.04	<0.01	<0.01	10.50	10.40	10.38	10.58	$\underline{10.35}$	10.30
4 X 4	<0.1	0.02	<0.01	<0.01	10.38	9.93	$\underline{9.78}$	10.05	9.83	9.72
4 X 5	<0.1	0.04	<0.01	<0.01	12.98	12.65	$\underline{12.32}$	12.65	12.40	12.25
4 X 6	<0.1	0.07	<0.01	<0.01	13.98	13.70	$\underline{13.35}$	13.58	13.38	13.22
4 X 7	<0.1	0.11	<0.01	0.01	15.95	15.78	15.48	15.75	$\underline{15.43}$	13.60
5 X 4	<0.1	0.08	<0.01	<0.01	16.10	15.52	15.45	15.38	$\underline{15.05}$	14.70
5 X 5	<0.1	0.16	0.02	0.01	19.35	18.80	18.68	18.38	$\underline{17.90}$	16.88
5 X 6	<0.1	0.30	0.02	0.03	22.35	22.08	21.62	21.83	$\underline{21.33}$	16.00
5 X 7	<0.1	0.40	0.04	0.04	24.25	23.58	23.45	23.73	$\underline{23.00}$	18.00
5 X 8	<0.1	0.75	0.05	0.05	27.68	27.03	$\underline{26.45}$	26.95	$\underline{26.45}$	-
5 X 9	0.1	1.10	0.04	0.07	30.63	30.05	29.12	29.83	$\underline{29.10}$	-
5 X 10	0.2	1.73	0.04	0.04	32.90	32.25	31.90	32.45	$\underline{31.68}$	-
6 X 6	0.4	0.72	0.08	0.12	32.53	31.13	30.40	30.95	$\underline{29.60}$	-
6 X 10	2.3	3.96	0.17	0.18	46.20	44.48	44.08	44.68	$\underline{43.45}$	-
10 X 6	45.5	38.87	0.83	1.16	89.28	83.03	85.45	81.78	$\underline{77.55}$	-
10 X 10	60.0	48.04	1.38	2.80	118.63	125.38	121.50	116.95	$\underline{110.75}$	-

(ACO-N) and Gre-C (ACO-C).

From the results in Tables 3, 4 we can see that the ACO-C manages to find better solutions than EXP and FB in the vast majority of cases. ACO-C always finds better solutions than FB, except in the case of small instances where both methods find the optimal solutions. In general ACO-C manages to outperform EXP, although in a few cases EXP manages to find slightly better results. It is important to note that the advantage of ACO-C is more pronounced in case of large problem instances reaching even 10% in case of $H_{max} = T+2$. Let us note that although Gre-C generally has a worse performance than FB and EXP, it manages to get better results in several problem sizes for $H_{max} = \infty$. ACO-N overall has a slightly better performance than FB but worse than EXP. This indicates once more, that the relocation of well-located containers is necessary for finding high quality solutions for uBRP.

Although the computational times for EXP, FB, ACO-N and ACO-C have not been acquired on the same machine, the advantage of the proposed ACO methods is evident. In case of the two largest problem instances the computational time of ACO-C was between 1.3 and 2.8 seconds, while for FB (EXP) it was between 45 (35) and 60 (48) seconds. When we compare the computational time of ACO-N and ACO-C the inclusion of well-located containers for potential relocations results in an increase in computational time between 50 and 200%.

7.4. Alternative objectives

In our final group of experiments we explore the robustness of the ACO in the sense of its application to other objective functions. ACO is applied to the rBRP with objective functions O_v, O_f given in Eqs. (42), (43). In case of all the tests we use $H_{max} = T + 2$. The parameter values for defining the objective functions are the same as in Schwarze and Voß (2015). To be more precise, the time needed for the crane to cross one stack is $t_s = 1.2s$. In case of O_f two settings are explored, in the first $O_{f,30}$ the pick-up/place-down operation has a high cost $t_{pp} = 30s$ and in the second $O_{f,5}$ a significantly lower one $t_{pp} = 5s$. In case of the objective function O_v, the parameter defining the time needed to move a container one tier was $t_r = 7.77s$. The height of the retrieval stack was $h_{out} = 1.5s$, and the maximal tier had the value $h_{max} = H_{max} + 1$.

The tests have been performed on the same group of data sets as in the previous subsections, with the constraint that only problem sizes are used for which all the optimal solutions are known from Schwarze and Voß (2015). We compare the average error, in seconds, E_{ACO} of the solution acquired using the ACO algorithm to the known optimal. With the intention of having a better understanding of the effect of the ACO, as in Schwarze and Voß (2015), we also consider the value of the objective function O_f (O_v) for the optimal solution of the rBRP where the objective function is the number of relocations O_r. The results of this comparison can be seen in Table 5. In it, we present the average error E_R of such solutions for the 40 problem instances of one size.

In this table we did not include computational times since they were very short. Note that the number of iterations of the ACO algorithm in case of the objective functions based on the crane operation time was around 5 times higher than in the case when the objective function is the number of relocations. This can be explained by the fact that the used heuristic function is designed for the latter, and as a consequence the convergence speed of the ACO algorithm is slower in case of the former. This also indicates the strong positive effect of using the pheromone matrix.

If we observe the performance of the ACO algorithm for the objective functions $O_{f,30}$, $O_{f,5}$ we can see the robustness of the approach. In case of $O_{f,30}$ the largest average error for ACO was only 1.7 seconds, which is less than 0.2% of relative error. This very good performance is closely related to the fact, as discussed in Schwarze and Voß (2015), that in only two problem instances the optimal solution for the objective O_f was not the optimal for the objective O_r. The significantly larger value of E_R is related to the fact that there are multiple optimal solutions for each instance and if one is selected at random it may not be a good solution for $O_{f,30}$. The proposed ACO approach proves to be effective in finding the best one out of all such solutions.

In case of $O_{f,5}$, ACO's largest error is 3.5 seconds, which is around 0.5% of relative error. Note that the relation between optimal solutions for the two objective functions O_r and $O_{f,5}$ is not as strong as in the case of $O_{f,5}$. To be more precise, the optimal solution for objective $O_{f,5}$ is also optimal for O_r in about 55% of the instances. This reflects that E_R has a relative error of 5-12%. A similar behavior can be seen for the objective function O_v. In this case the largest error of ACO was 20 seconds, and the relative one is slightly more than 1%. From the performance of ACO for $O_{f,5}$ and O_v we can see that the use of the pheromone matrix manages, to a large extent, to overcome the problem of having a relatively poor heuristic function and lower bounds.

Table 5: Comparison of the average solution quality for different objective functions in case $H_{max} = T + 2$.

T X S	$O_{f,30}$			$O_{f,5}$			O_v		
	OPT	E_R	E_{ACO}	OPT	E_R	E_{ACO}	OPT	E_R	E_{ACO}
3 X 3	476.4	2.3	0.0	126.4	2.3	0.0	904.0	16.3	1.8
3 X 4	633.6	7.9	0.1	178.4	8.7	0.0	1187.0	40.2	6.2
3 X 5	789.2	16.3	0.1	236.6	18.3	0.3	1460.8	61.6	6.8
3 X 6	968.2	22.4	0.1	303.2	27.4	0.8	1758.4	89.8	12.4
3 X 7	1137.0	35.8	0.0	375.2	40.7	0.9	2043.4	124.7	18.3
3 X 8	1331.3	43.3	0.3	455.1	53.2	1.9	2362.6	147.2	25.5
4 X 4	911.9	10.1	0.0	255.5	11.5	0.2	1928.1	67.6	10.7
4 X 5	1170.2	19.8	0.0	343.0	23.3	1.3	2424.6	131.6	20.2
4 X 6	1385.4	31.4	0.4	428.5	37.8	2.2	-	-	-
4 X 7	1647.7	53.0	0.8	532.8	64.8	3.5	-	-	-
5 X 4	1230.0	16.1	1.7	343.5	17.0	0.6	-	-	-

It is important to note that the performance of the proposed ACO method could be significantly improved by developing and using a more efficient heuristic function and lower bounds for the BRP with objective functions based on crane working times. In this paper we did not focus on this issue since our main objective was on presenting the ACO method. On the other hand, we also wanted to show that this approach, even in a simple form that is easy to implement, can achieve very good results.

7.5. Summary

In this section we have presented results of extensive computational experiments of the proposed greedy and ACO algorithms on the rBRP and uBRP with different objective functions. Several observations can be made. In case of all the variations of the BRP the use of ACO proves to be very efficient in case of both computational cost and solution quality. To be more precise, it manages to consistently find better or equivalent quality solutions as state-of-the-art methods in several different settings. In case of large problem instances, it finds such solutions at a fraction of the time (generally 20-30 times faster) when compared to competing methods. We have also experienced that the additional calculations for maintaining and using the pheromone matrix represent a small part of the overall calculations. This indicates that there is potential of using the same ACO mechanism with other more complex heuristic methods.

Another observation is that even by using the standard $MinMax$ heuristic, in case of the greedy algorithm for the uBRP, a significant improvement on the quality of found solutions can be achieved by simple extending the candidate list of relocations. We have shown that if relocations of well-located containers are allowed in some specific cases the quality of found solutions can be further improved for a reasonable increase in computational cost.

8. Conclusion

In this paper we have presented an ACO method for solving the BRP in case of distinct due dates. The method has been applied for both the restricted and unrestricted version of the problem. In developing the ACO algorithm, a new, simple to implement, heuristic function has been proposed for solving the uBRP that manages to outperform similar existing approaches. Further, a new formulation of the standard greedy algorithm for the rBRP is introduced, which has the advantage that it can be directly applied to the uBRP. The proposed greedy algorithm has been extended towards ACO. In developing this approach a novel concept has been introduced for defining the pheromone matrix that only contains partial information about the problem. Our computational results show that the proposed ACO algorithm manages to outperform the current state-of-the-art in both computational cost and quality of found solutions. Further, we have shown that the proposed ACO algorithm is highly robust in the sense that it can be extended to alternative objective functions for the BRP.

In the future we plan to extend this work by adapting the proposed ACO algorithm to the closely related premarshalling problem and to other variations of the BRP. Another direction of future research is enhancing the performance of the proposed algorithm by incorporating a multi-colony approach or by incorporating a local search, similar to the work in Gambardella et al. (2012).

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., 2017. Blocks relocation problem. URL: `https://github.com/marcocaserta/BRP`.

Caserta, M., Schwarze, S., Voß, S., 2009. A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. Lecture Notes in Computer Science 5482, 37–48.

Caserta, M., Schwarze, S., Voß, S., 2011a. Container rehandling at maritime container terminals, in: Böse, J.W. (Ed.), Handbook of Terminal Planning. 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, 96–104.

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

Dorigo, M., Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1, 53–66.

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, 327–343.

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.

Forster, F., Bortfeldt, A., 2012. A tree search procedure for the container relocation problem. Computers & Operations Research 39, 299 – 309.

Gambardella, L., Montemanni, R., Weyland, D., 2012. Coupling ant colony systems with strong local searches. European Journal of Operational Research 220, 831 – 843.

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.

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

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

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

Lee, Y., Kang, J., Ryu, K., Kim, K., 2005. Optimization of container load sequencing by a hybrid of ant colony optimization and tabu search. Lecture Notes in Computer Science 3611, 433–433.

Lee, Y., Lee, Y.J., 2010. A heuristic for retrieving containers from a yard. Computers & 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.

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.

Ndiaye, N.F., Yassine, A., Diarrassouba, I., 2014. A hybrid ant colony and genetic algorithm to solve the container stacking problem at seaport terminal, in: 2014 International Conference on Advanced Logistics and Transport (ICALT), pp. 247–252.

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, 1947–1966.

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, 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.

Shishvan, M.S., Sattarvand, J., 2015. Long term production planning of open pit mines by ant colony optimization. European Journal of Operational Research 240, 825 – 836.

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: 19th International Conference on Intelligent Transportation Systems (ITSC), IEEE. pp. 933–938.

Sreeja, N., Sankar, A., 2015. Ant colony optimization based binary search for efficient point pattern matching in images. European Journal of Operational Research 246, 154 – 169.

Steenken, D., Voß, S., Stahlbock, R., 2004. Container terminal operation and operations research - a classification and literature review. OR Spectrum 26, 3–49.

Stützle, T., Hoos, H.H., 2000. MAX–MIN ant system. Future Generation Computer Systems 16, 889–914.

Sun, B., Sun, J., Liu, F., Yang, P., Han, M., Feng, M., 2010. On robust discrete berth allocation based on ant colony algorithm, in: Proceedings of the 29th Chinese Control Conference, pp. 1727–1732.

Tanaka, S., Takii, K., 2016. A faster branch-and-bound algorithm for the block relocation problem. IEEE Transactions on Automation Science and Engineering 13, 181–190.

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

Tuba, M., Jovanovic, R., 2009. An analysis of different variations of ant colony optimization to the minimum weight vertex cover problem. WSEAS Transactions on Information Science and Applications 6, 936–945.

Tus, A., Rendl, A., Raidl, G.R., 2015. Metaheuristics for the two-dimensional container pre-marshalling problem, in: International Conference on Learning and Intelligent Optimization, Springer. pp. 186–201.

Ünlüyurt, T., Aydın, C., 2012. Improved rehandling strategies for the container retrieval process. Journal of Advanced Transportation 46, 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, 699–713.

Wilmsmeier, G., Froese, J., Zotz, A., Meyer, A., 2014. Energy consumption and efficiency: Emerging challenges from reefer trade in South American container terminals. FAL Bulletin 329.

Wu, K.C., Ting, C.J., 2012. A beam search algorithm for minimizing reshuffle operations at container yards, in: Proceedings of the 2010 International Conference on Logistics and Maritime Systems, Busan, Korea.

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, 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, 48–62.

Zhang, C., 2000. Resource Planning in Container Storage Yard. Ph.D. thesis. Department of Industrial Engineering, Hong Kong University of Science and Technology.

Zhang, D., Du, L., 2011. Hybrid ant colony optimization based on genetic algorithm for container loading problem, in: 2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR), pp. 10–14.

Zhu, M., Fan, X., He, Q., 2010. A heuristic approach for transportation planning optimization in container yard, in: 2010 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 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, 710–722.