Computers & Industrial Engineering 75 (2014) 79–86

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A chain heuristic for the Blocks Relocation Problem

Raka Jovanovic [a,b,*], Stefan Voß [c]

[a] Qatar Environment and Energy Research Institute (QEERI), PO Box 5825, Doha, Qatar
[b] Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, Zemun, Serbia
[c] Institute of Information Systems, University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany

ARTICLE INFO

Article history:
Received 8 January 2013
Received in revised form 13 June 2014
Accepted 14 June 2014
Available online 23 June 2014

Keywords:
Blocks relocation
Logistics
Heuristics

ABSTRACT

In the Blocks Relocation Problem (BRP) one is given a block retrieval sequence and is concerned with determining a relocation pattern minimizing the total number of moves required to enforce the given retrieval sequence. The importance of the BRP has been constantly growing in recent years, as a consequence of its close connection with the operations inside of a container terminal. Due to the complexity of the BRP, a large number of methods has been developed for finding near optimal solutions. These methods can be divided in two main categories greedy heuristics and more complex methods. The latter achieve results of higher quality, but at the cost of very long execution times. In many cases, this increased calculation time is not an option, and the fast heuristic methods need to be used. Greedy heuristic approaches, in general, apply the heuristic based only on the properties of the block that is being relocated and the current state of the bay. In this paper we propose a new heuristic approach in which when deciding where to relocate a block we also take into account the properties of the block that will be moved next. This idea is illustrated by improving the Min–Max heuristic for the BRP. We compare the new heuristic to several existing methods of this type, and show the effectiveness of our improvements. The tests have been conducted on a wide range of sizes of container bays, using standard test data sets.

1. Introduction

With the globalization of the world economy, container transport is becoming of great importance. Container terminals are gigantic logistic centers, that are used for the distribution of containers. More precisely, they can be seen as temporary storage points, that make possible unloading operations from very large transport vessels and loading operations onto smaller vehicles, like trains or trucks, for further distribution, but also the same process just in the opposite direction. Due to the fierce competition of the global market, the efficiency of container terminals is of utmost importance. One way of improving their productivity is by using new technologies like automated guided vehicles (AGVs), systems based on automated lift vehicles (ALVs), more efficient cranes, etc. (Duinkerken, Dekker, Kurstjens, Ottjes, & Dellaert, 2006; Stahlbock & Voß, 2008). Increasing the effectiveness of the container terminal operations can also be done by optimizing the way in which such operations are carried out using existing equipment.

One of the most important aspects of a storage system is the time needed for container loading to transport vehicles and vessels. This is due to the fact that the faster the retrieval operations are completed, the sooner expensive and scarce resources (e.g., trucks, staff, cranes, etc.) can be used for new jobs, or in other words they are used more efficiently. Furthermore, terminals usually have a limited amount of storage space, which has the consequence that containers are piled up at the container yard in such a manner to increase the space utilization. Block stacking is the most common way for container storage at container yards (Kim & Hong, 2006). The problem with block stacking is that only the top container can be retrieved from each stack, while containers often need to be loaded to transport vehicles in a certain order. This order (approximate) is usually known before the loading process starts, but it does not generally correspond to the state of the container yard (Fig. 1). This means that not only do containers need to be moved from the container block to the transport vehicle but they will also have to be relocated within the container block to make retrieval in the specified order possible. The movement of containers is time consuming and the number of relocations should be minimized. There are several approaches to solving this practical problem which have been formalized in the form of the Blocks Relocation Problem (BRP), the Re-Marshalling Problem

* Corresponding author at: Qatar Environment and Energy Research Institute (QEERI), PO Box 5825, Doha, Qatar. Tel.: +974 66513631.
E-mail addresses: rakabog@yahoo.com (R. Jovanovic), stefan.voss@uni-hamburg.de (S. Voß).

http://dx.doi.org/10.1016/j.cie.2014.06.010
0360-8352/

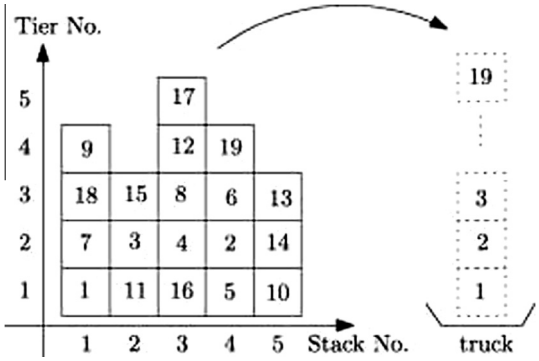

Fig. 1. Container bay with 19 containers in five stacks; containers need to be retrieved in increasing order of their numbering. Image taken from Caserta et al. (2012).

(RMP), i.e. intra-block marshalling and the Pre-Marshalling Problem (PMP) (Caserta, Schwarze, & Voß, 2011a).

The BRP, which we shall consider in this paper, is defined in the following way. First we will consider the problem setup with some simplified assumptions as they are consistently used in literature (Kim & Hong, 2006):

- All blocks (containers) are of the same size.
- The container bay will be viewed as a two dimensional stacking area, with W stacks, for which a maximal height (number of tiers) H is given.
- The initial configuration of the container bay and the retrieval sequence are known in advance (Fig. 1).
- The reshuffle operations (movement of containers within the bay) are only allowed while a target container needs to be retrieved, which means no pre-marshalling.
- Only blocks from the top of a stack can be accessed.
- Blocks can only be placed either on top of another block, or on the ground (tier 0).
- When a block is retrieved, it is removed from the container bay.

The problem is to minimize the number of moves needed for retrieving a given block sequence.

It has been shown that this problem is NP-hard (Caserta, Schwarze, & Voß, 2012). There have been several directions for solving this problem both for finding optimal and approximate solutions. In their article, Kim and Hong (2006) present a branch and bound method for finding the optimal solution of the BRP. In the same article, a heuristic method is presented for finding an approximate solution. The problem has been presented by a binary linear programming model, and solved using a heuristic method (Caserta et al., 2012; Caserta, Schwarze, & Voß, 2009). Another heuristic approach, in the form of the beam search algorithm, is given by Wu and Ting (2010). All the presented approximate algorithms are deterministic. Though, it is a well known fact that for NP-hard problems, in many cases probabilistic algorithms can achieve results of higher quality for a lower computational time. A nondeterministic algorithm has been implemented for the BRP using the corridor method paradigm (Caserta, Voß, & Sniedovich, 2011b). A population based approach is presented in (Hussein & Petering, 2012) where a variation of the BRP called block relocation problem with weights (BRP-W) has successfully been solved using a genetic algorithm. A very interesting version of the BRP is solved by Lee and Lee (2010) in which more than one bay is considered. This effectively transforms the initial problem into a three dimensional one which, as a consequence, is of much greater size (the authors mention considering problem instances with more than 700 containers).

Although the more complex methods can achieve results of higher quality than greedy heuristics, it is at a high computational cost. In many real life applications the extra calculation time is not available and it is necessary to use some fast heuristic approach. On the other hand heuristics used in greedy algorithms are often a basis for developing more complex methods. In this article we present a new approach to using a greedy heuristic for the BRP. The main idea is to use a heuristic function that is dependent on properties of more than one block, when deciding where to relocate a single block. We illustrate the effectiveness of this method by applying the new chain heuristic approach developed in this paper to the greedy algorithm given in Caserta et al. (2011b). We also present a simple improvement for the heuristic given in Caserta et al. (2011b).

We compare the new heuristic approach to several standard heuristics used on the BRP. We show that the new method achieves the best results of all the tested methods on some standard benchmark data sets. We also show that the new approach has a neglectable effect on the overall calculation time.

This article is organized as follows. In the next section we give an overview of greedy heuristics for the BRP, and introduce one simple improvement. In the third section, we explain the concept of a chain heuristic. In the fourth section we provide a comparison of different heuristics for the BRP. In the final section we give some concluding remarks.

2. Related work

There have been several greedy algorithms developed for solving the BRP. The general idea of this approach is to move a block that is blocking another block that needs to be retrieved, to a new stack. This stack is selected among the set of all stacks by some heuristic function that measures their desirability. Several different heuristics have been developed, for which we give a short overview.

The most basic idea for a heuristic is to move the blocking block to a new stack that has the lowest number of tiers filled with containers. This approach has also been called the Lowest Position heuristic (TLP) (Zhang, 2000). It is illustrated by the following equation:

$$s^* = f_{TLP}(r, Bay_n) = argmin_{i\in\{1,\dots,W\}\setminus\{s\}}\, t(i) \tag{1}$$

In Eq. (1), s is the index of the stack from which the block r is moved, s^* is the index of the stack to which the block will be moved to, $t(i)$ gives us the number of tiers filled with containers for stack i. The variable Bay_n gives the state of the bay at step n, or, in other words, what are the elements at each stack. We wish to point out that the value of s is specified by the properties of block r, and is used as a separate variable simply for convenience. The effect of using TLP is that all of the stacks will have a similar number of tiers, so the average number of relocations would stay low. The idea behind this heuristic is to avoid extreme cases where a large number of blocks needs to be moved from a stack with many tiers when a block with high priority is blocked.

Murty et al. (2005) have proposed the Reshuffle Index heuristic (RI) for the BRP. In this approach, the blocking block will be moved to the stack in which it blocks the lowest number of blocks. More precisely, it will be moved to the stack which has the lowest number of blocks that have a higher priority than the block being moved. This heuristic can be illustrated by the following equation:

$$s^* = f_{RI}(r, Bay_n) = argmin_{i\in\{1,\dots,W\}\setminus\{s\}}\, RI(r, i) \tag{2}$$

In Eq. (2), $RI(r, i)$ gives the number of blocks in stack i that have a higher priority than r. In this way it is expected that the overall number of reshuffles is lowered since every time a block is put over

another with a higher priority, extra reshuffles need to be done. In the original implementation of this method, if a tie occurs between stacks, the choice would be made arbitrarily. Wu and Ting present an improvement of this method by giving more consideration to how ties are resolved (Wu & Ting, 2010). In the case some stacks have the same RI, the tie is broken by selecting the stack that has the lowest maximal priority. In this way new relocations in the near future are avoided. It is interesting to mention that just by a different way of resolving ties a significant improvement has been achieved.

Caserta et al. (2011b) have presented a very efficient heuristic that only takes into account the maximal priority of a block in each stack. A very similar heuristic is presented in Ünlüyurt and Aydin (2012). The proposed heuristic is called the Min–Max heuristic. It has a different way of choosing the stack to which the block will be moved to, depending if it creates a new deadlock or not. (Whenever a pair of blocks is located in the same stack, and the priority of the lower block is higher than the priority of the upper block, we say that this pair of blocks forms a deadlock.) It first attempts to move the block to a stack without creating new deadlocks. If the block creates no new deadlocks it is moved to the stack that has the highest maximal priority. In case the block movement has to create a new deadlock it will be moved to the stack that has the lowest maximal priority. The idea behind this heuristic is that stacks with lower values of maximal priorities are more valuable. This is because a greater number of blocks can be added to that stack without creating new deadlocks. On the other hand, for blocks that have low priorities it is best to put them somewhere where they will be "out of the way" for as long as possible.

We shall present this heuristic in a slightly different form than given in (Caserta et al., 2011b). First we shall define $p(i)$ as the highest priority of a block in stack i. In case a stack i is empty $p(i) = N + 1$, where N represents the lowest priority of a block or in other words the number of blocks in the initial bay. Next we define

$$d(i,r) = p(i) - r \tag{3}$$

Function $d(i,r)$ gives us the difference in the maximal priority of stack i and the priority r of the block being moved. Using function $d(i,r)$ we can define the Min–Max heuristic using the following equation:

$$s^* = f_{MinMax}(r, Bay_n) = \begin{cases} argmin_{i\in\{1,\dots,W\}\setminus\{s\}}\, d(i,r), & \exists d(i,r) > 0 \\ argmax_{i\in\{1,\dots,W\}\setminus\{s\}}\, d(i,r), & \forall d(i,r) < 0 \end{cases} \tag{4}$$

We wish to mention that in Eq. (4), if $d(i,r) < 0$ it means that by moving block r to stack i a new deadlock is created.

We propose a simple improvement to the Min–Max heuristic by taking into account if a stack will become filled (maximal level reached). In the case a new deadlock is being created, the Min–Max heuristic gives us the stack that has the lowest priority. If the chosen stack has become filled, we have effectively lost the possibility of adding new blocks to that stack. The selected stack can receive blocks with the lowest priority without creating new deadlocks. This practically means that in this case the Min–Max heuristic has given us the worst possible solution. In the case of creating a new deadlock and moving a block to a stack that will reach the maximal number of tiers, this problem can be avoided by using a simple correction. In this case, instead of using $p(i)$ we use

$$p^*(i) = -N - p(i) \tag{5}$$

in Eq. (4) when calculating $argmax_{i\in\{1,\dots,W\}\setminus\{s\}} d(i,r)$. The effect of this modification can be seen in Figs. 2 and 3.

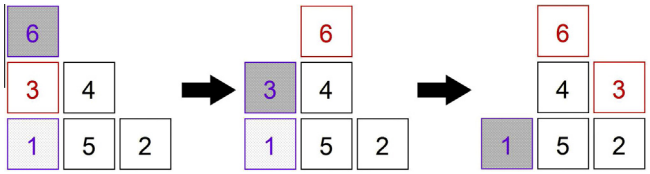

Fig. 2. Illustration of two steps of the Min–Max heuristic.

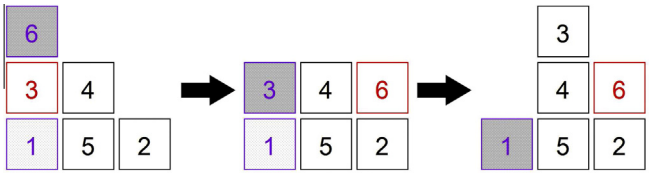

Fig. 3. Illustration of two steps of the Min–Max heuristic improved with checks for stacks getting full.

3. Chain heuristic

All the presented greedy heuristic approaches apply the heuristic based only on the block that is being relocated and the current state of the bay.

More precisely, if at some step n, block r_n is being relocated, the heuristic function depends on the state of the container bay Bay_n, and the properties of the block r_n. Let this function be denoted as $f(r_n, Bay_n)$. One direction of improving greedy heuristics for the BRP is by relocating a group of blocks instead of only one. We shall call this an extended heuristic. It is obvious that the optimal relocation of a group of k blocks may be better than k consecutive optimal relocations of one block. The problem is that the number of possible relocations of a group of k blocks grows exponentially, and it will be W^k. The calculation time for an algorithm based on an extended heuristic will be much larger than the basic heuristic. This calculation time can even come close to the one needed for more complex methods, if a large value of k is used. This way the main advantage, the speed, of the heuristic method has been lost and the achieved results are still not as good as when using more complex methods. A possible way for improvement and extension is to look ahead within heuristic search (Duin & Voß, 1999). An interesting look-ahead approach to the BRP is given by Petering and Hussein (2013), with their LA-N heuristic. In this method they also take into account the state of stacks that contain blocks that will be retrieved after the current block. In the LA-N heuristic they allow blocks from these stacks also to be relocated, which is a variation of the BRP since it has some types of pre-marshalling allowed.

In this section we present a new chain heuristic approach that will have at least some of the advantages of the extended heuristic but will not suffer from the extreme increase in calculation time. The idea for this approach comes from the fact that even when using an extended heuristic it is still just a heuristic that "guesses" what is a good relocation strategy. The extended heuristics are, in most cases, defined by some combination (frequently a simple sum) of the heuristic for single blocks. The desired effect of an extended heuristic is to avoid having blocks, that are relocated earlier, use stacks that are more suitable for blocks that will be relocated later.

We propose a simple method that can approximate the effect of an extended heuristic. It is obvious that at every step n, we know which block r_n is being relocated, but we also know block r_{n+1} which will be relocated at step $n+1$. Block r_{n+1} can be located at one of the following two locations:

- In the same stack as r_n just under it.
- Blocking the next block that needs to be removed.

It is important that these two blocks are independent of the reshuffle operations, except the extreme case that block r_n will have to be moved two times in a row. If we are using a simple heuristic, we first calculate $f(r_n, Bay_n)$ to get the stack where r_n will be relocated. Next, we calculate $f(r_{n+1}, Bay_{n+1})$ to see where we should relocate block r_{n+1}.

In the case we are using an extended heuristic, we shall calculate $f_e((r_n, r_{n+1}), Bay_n)$ to see how the two blocks should be relocated. At the next step of the algorithm we shall calculate where the next two blocks (r_{n+2}, r_{n+3}) should be relocated.

In the new greedy method, instead of using a heuristic function $f(r_n, Bay_n)$ we would use $f((r_n, r_{n+1}), Bay_n)$ to decide where to relocate block r_n. In the following step we would use $f((r_{n+1}, r_{n+2}), Bay_{n+1})$ to decide where to relocate block r_{n+1}. As we have previously mentioned the goal of an extended heuristic is to avoid the situation where blocks that are relocated earlier will use stacks that are more suitable for blocks that are being relocated later. The new heuristic function only needs to give us an answer to a simple question of what is better:

- Get the relocation for block r_n based on the best value of f for the current state of the bay Bay_n, and then get the relocation for block r_{n+1} using f for the new state of the bay Bay_{n+1}.
- Get the relocation for block r_{n+1} based on the best value of f for the current state of the bay Bay_n, and then get the relocation for block r_n based on the current state of the bay Bay_n with the constraint that the move for r_{n+1} will still be possible. The constraint is in practice very simple, one can not put block r_n on the stack where r_{n+1} has been placed.

It is relatively easy to define $f((r_n, r_{n+1}), Bay_n)$, using $f(r_n, Bay_n)$ and $f(r_{n+1}, Bay_n)$ if f corresponds to the Min–Max heuristic. To do this we will first define an extended version of f_{MinMax}

$$f^*_{MinMax}(r, s_e, Bay_n) = \begin{cases} argmin_{i\in\{1,\dots,W\}\setminus\{s,s_e\}}\, d(i,r), & \exists d(i,r) > 0 \\ argmax_{i\in\{1,\dots,W\}\setminus\{s,s_e\}}\, d(i,r), & \forall d(i,r) < 0 \end{cases} \tag{6}$$

The extended function f^*_{MinMax}, is the same as the original except that an extra stack s_e can be excluded during the search for minimal, or maximal values of $d(i,r)$.

Next we need to evaluate the desirability of the order of relocations, or in other words what would be the impact to first relocate block r_n and then r_{n+1}, or in reverse. We first give the equations connected with the direct order of relocations

$$s^*_D = f_{MinMax}(r_n, Bay_n), \quad D_1 = d(s^*_D, r_n) \tag{7}$$

$$s^{**}_D = f_{MinMax}(r_{n+1}, Bay_{n+1}), \quad D_2 = d(s^{**}_D, r_{n+1}) \tag{8}$$

In Eqs. (7), (8), D_1 and D_2 give the desirability of moving r_n and then r_{n+1} using the Min–Max heuristics.

$$s^{**}_R = f_{MinMax}(r_{n+1}, Bay_n), \quad R_2 = d(s^{**}_R, r_{n+1}) \tag{9}$$

$$s^*_R = f^*_{MinMax}(r_n, s^{**}_r, Bay_n), \quad R_1 = d(s^*_R, r_n) \tag{10}$$

Similar, in Eqs. (9), (10), R_1 and R_2 give the desirability of moving $r_n + 1$ and then r_n using the Min–Max heuristics. Finally, Eq. (11) provides the index s^* of the stack to which block r_n will be moved to.

$$s^* = \begin{cases} s^{**}_R, & (R_2 < D_1) \wedge (R_2, D_1 > 0) \wedge (R_1 D_2 > 0) \\ s^*_D, & \text{otherwise} \end{cases} \tag{11}$$

As previously mentioned, our goal when using a chain heuristic, is to decide what is better, to first move r_n and then r_{n+1} or in reverse order. Eq. (11) gives an answer to that question. The reverse order will only be used if an improvement can be achieved, more precisely if all of the following criteria are satisfied:

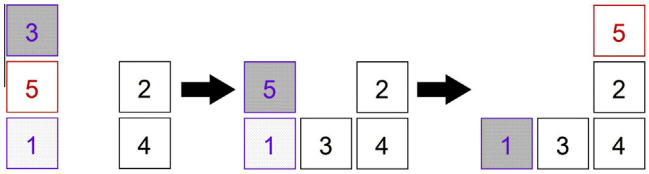

Fig. 4. Illustration of two steps of the Min–Max heuristic.

- $R_2, D_1 > 0$, moves specified by R_2, D_1 do not create new deadlocks.
- $R_1 D_2 > 0$, moves specified by R_1, D_2 both create deadlocks, or both of them do not.
- $R_2 < D_1$, block r_{n+1} is better relocated to stack s^{**}_R, then block r_n is relocated to stack s^*_D

We wish to point out that if Min–Max is used as the basic heuristic, we do not need to calculate the chain heuristic at every step. Because of the way the Min–Max heuristic has been defined, it is only possible for block r_n to use a stack that is more suitable for r_{n+1} if $p(r_n) < p(r_{n+1})$. On the other hand, the values for R_1, R_2, D_1, D_2 can be calculated jointly in a very efficient way. These two facts can be exploited when developing software that implements this method.

The effect of using the chain heuristic is illustrated in Figs. 4 and 5. Let us point out that when using a chain heuristic, the level of "lookahead" can be higher than an extended heuristic of length 2 (or even higher in specific cases). An example of this effect can be an array of blocks that are in ascending order by priority, that are being relocated to an empty stack.

For better understanding of the chain heuristic, we give the following pseudo-code[1]:

```
i = 1
n = 1
while i < N + 1 do
  if CanRemoveBlock (i) then
    RetrieveBlock (i)
    i = i + 1
  else
    GetBlocks r_n, r_{n+1} for i
    s* = f_MinMax(r_n, Bay_n)
    if p(r_n) < p(r_{n+1}) then
      Calculate R_1, R_2, D_1, D_2
      if (R_2 < D_1) ∧ (R_2, D_1 > 0) ∨ (R_1 D_2 > 0) then
        s* = f_MinMax(r_{n+1}, Bay_n)
      end if
    end if
    MoveTo (r_n, s*)
  end if
  n = n + 1
end while
```

4. Experimental results

In this section we give a comparison of the newly presented chain heuristic with other heuristic methods. The test data used in the comparison is taken from Wu and Ting (2010). The test states of the container bay have been generated using guidelines

[1] We assume that the functions within the pseudo-code are somewhat self-explanatory. For instance, if block i with highest priority is available with no other block on top, CanRemoveBlock (i) is true and block i can be retrieved according to RetrieveBlock (i).

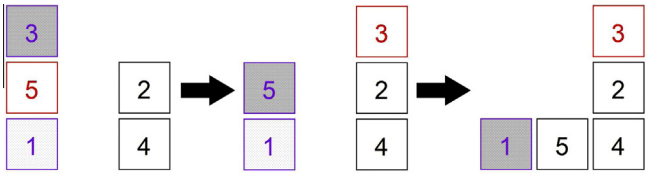

Fig. 5. Illustration of two steps of the Min–Max using the chain heuristic approach.

given in Wan, Liu, and Tsai (2009). In all the generated test cases the container bay is almost full. We compare the performance for a wide range of problem sizes, for 3–12 stacks and 3–12 tiers. For each of the test sizes, 40 different problem instances are considered. We present the average results for each test size in Tables 1, 3, and 5. In the tables the values for the approaches of Murty et al., Wu and Ting (their improvement of RI) and beam search are taken from Wu and Ting (2010). The algorithm of Wu and Ting is the best heuristic presented in Wu and Ting (2010), and the original specification of the RI heuristic is given in Murty et al. (2005). The values for Caserta et al. (Min–Max) correspond to the heuristic algorithm presented in Caserta et al. (2012), in our implementation. The values in columns Chain and Chain F show the effectiveness of the two improvements presented in the previous section. All of the algorithms have been implemented in C# using Microsoft Visual Studio 2012. The calculations have been done on a machine with Intel(R) Core(TM) i7-2630 QM CPU 2.00 GHz, 4 GB of DDR3-1333 RAM, running on Microsoft Windows 7 Home Premium 64-bit.

In the Tables 1 and 3 we include the values of Wu and Ting's beam search as a lower bound for the problem instances. This

Table 2
The range and standard deviation of the level of improvement for individual problem instances for small sized bays given in percent of the Chain and Chain F heuristics compared to the Min–Max heuristic.

Tiers × stacks	Chain			Chain F		
Stacks	Imp.	Range	Stdev.	Imp.	Range	Stdev.
3 × 3	0.00	(0, 0)	0.00	0.26	(0, 20)	10.73
4	0.00	(0, 0)	1.73	−1.20	(−13, 25)	10.68
5	0.00	(0, 0)	0.00	2.05	(−33, 36)	13.51
6	1.34	(0, 22)	4.41	3.64	(−11, 25)	9.39
7	1.87	(−8, 19)	5.03	0.65	(−20, 19)	9.20
8	2.26	(−5, 24)	5.76	3.10	(−18, 31)	9.27
3 × 4	0.00	(0, 0)	0.00	0.00	(0, 0)	0
4	0.19	(0, 8)	1.20	1.27	(−11, 29)	6.18
5	0.56	(−13, 10)	3.54	−1.17	(−25, 39)	10.57
6	0.57	(−7, 17)	3.37	1.20	(−19, 23)	6.59
7	1.35	(−8, 15)	4.41	1.74	(−25, 33)	10.33
8	1.79	(−9, 13)	4.35	1.51	(−31, 30)	13.91
3 × 5	0.00	(0, 0)	0.00	−0.67	(−14, 0)	2.93
4	0.36	(−6, 13)	2.56	−0.18	(−19, 13)	6.13
5	0.54	(−7, 16)	3.35	1.78	(−21, 21)	8.22
6	1.68	(−7, 21)	5.19	0.93	(−21, 24)	10.32
7	2.07	(−9, 20)	5.84	0.75	(−17, 17)	8.78
8	0.95	(−19, 11)	4.96	1.73	(−19, 24)	10.79
3 × 6	0.00	(0, 0)	0.00	−0.32	(−7, 9)	4.95
4	0.61	(−7, 14)	2.82	0.21	(−12, 23)	4.82
5	0.67	(−7, 9)	2.89	0.21	(−7, 14)	5.88
6	0.73	(−11, 11)	3.94	1.00	(−13, 26)	7.89
7	2.51	(−11, 17)	5.91	3.27	(−13, 30)	9.48
8	2.37	(−9, 11)	5.29	3.47	(−13, 18)	7.46

The higher level of improvement for the two heuristics has been underlined.

Table 1
Comparison of different heuristics for the BRP on small size instances. Values for Murty et al. Wu/Ting, Beam Search (Wu/Ting) are taken from Wu and Ting (2010). Caserta et al. is our implementation of the Min–Max algorithm. Chain, Chain F are the Min–Max heuristic combined with only the chain heuristic, or with both improvements. In column 'Beam search' bold *italic* values are equal to known optimal solutions.

Tiers × stacks	Murty et al.	Wu/Ting	Caserta et al.	Chain	Chain F	Beam search (Wu/Ting)
3 × 3	3.42	3.4	3.42	3.42	3.38	***3.38***
4	6.10	5.95	5.82	5.82	5.95	***5.67***
5	9.80	9.45	9.10	9.10	8.70	***8.40***
6	13.60	13.20	12.97	12.77	12.30	***11.50***
7	18.10	17.30	16.62	16.25	16.40	***15.00***
8	24.10	22.90	22.02	21.52	21.20	***18.60***
3 × 4	5.03	4.92	4.95	4.95	4.95	***4.85***
4	9.05	8.80	8.75	8.72	8.57	***8.43***
5	14.50	13.70	13.12	13.02	13.17	***12.20***
6	19.10	17.90	17.15	17.05	16.92	***15.60***
7	28.90	27.60	26.37	25.97	25.62	22.60
8	37.90	34.60	33.67	33.05	32.82	27.90
3 × 5	5.90	5.75	5.75	5.75	5.80	***5.75***
4	12.20	11.80	11.40	11.35	11.40	***11.00***
5	18.10	17.40	17.07	16.95	16.65	***15.60***
6	25.60	24.10	23.92	23.52	23.57	21.10
7	36.30	33.70	31.92	31.15	31.42	27.80
8	49.80	44.50	43.07	42.57	42.00	36.40
3 × 6	7.92	7.80	7.72	7.72	7.85	***7.65***
4	13.20	12.80	12.67	12.55	12.60	***12.00***
5	22.60	21.40	20.60	20.42	20.52	19.30
6	32.60	30.10	29.02	28.77	28.57	26.10
7	45.50	41.70	40.42	39.22	38.65	34.90
8	57.20	53.90	52.20	50.87	50.25	43.20
Average						
Value	21.52	20.19	19.57	19.27	19.13	17.29
$100 * \frac{X-BS}{BS}$	24.48	16.80	13.20	11.45	10.68	0

The values of the best average solutions have been underlined.

Table 3
Comparison of different heuristics for BRP on medium size instances. Values for Murty et al. Wu/Ting, Beam Search (Wu/Ting) are taken from Wu and Ting (Wu & Ting, 2010). Caserta et al. is our implementation of the Min–Max algorithm. Chain, Chain F are the Min–Max heuristic combined with only the chain heuristic, or with both improvements. In column 'Beam search' bold *italic* values are equal to known optimal solutions.

Tiers × stacks	Murty et al.	Wu/Ting	Caserta et al.	Chain	Chain F	Beam search (Wu/Ting)
3 × 7	10.10	9.88	9.02	9.05	9.17	***8.95***
4	20.10	18.80	16.35	16.45	16.07	15.50
5	30.90	28.30	23.15	22.92	22.70	21.40
6	45.00	41.30	34.40	34.07	34.07	31.00
7	59.80	54.50	45.62	44.92	44.27	39.40
8	76.60	68.70	59.02	57.97	56.32	49.90
3 × 8	11.90	11.70	9.87	9.87	9.75	***9.72***
4	20.90	20.00	18.72	18.52	18.52	18.00
5	34.70	32.10	27.67	27.37	26.82	25.40
6	50.60	46.50	40.47	39.65	38.90	36.00
7	68.40	61.60	52.10	51.15	50.82	45.10
8	88.90	79.50	66.37	65.12	64.55	57.30
3 × 9	12.40	12.10	11.62	11.60	11.57	11.40
4	24.70	23.60	19.85	19.72	19.87	19.10
5	37.50	35.10	31.00	30.35	30.55	28.70
6	55.70	50.20	44.90	44.50	43.30	40.00
7	76.70	68.80	59.87	57.92	57.35	51.50
8	97.90	87.40	76.32	75.05	73.50	66.00
3 × 10	14.60	14.40	12.02	12.02	11.97	11.90
4	26.20	24.70	23.60	23.32	23.25	22.40
5	41.20	38.50	34.50	33.80	33.45	31.80
6	60.20	54.40	48.80	47.77	47.45	44.00
7	85.10	75.80	65.72	64.37	63.60	57.50
8	111.00	96.70	83.90	82.02	79.35	71.20
Average						
Value	48.38	43.94	38.11	37.47	36.96	33.88
$100 * \frac{X-BS}{BS}$	42.78	29.68	12.51	10.61	9.10	0

The values of the best average solutions have been underlined.

method has in many cases achieved the optimal solutions. We would like to mention that a direct comparison between the heuristic methods and the beam search is not of importance due to the great difference in execution time. In our test we did not include the calculation times for the heuristics, due to the fact that they are all very small, as mentioned in Wu and Ting (2010). On the other hand, from the description of the heuristics it is obvious that they have a very similar execution time.

We first observe the results for small and medium size instances. We notice that the Min–Max heuristic gets significantly better results than Wu/Ting's. The average number of reshuffle operations for Wu/Ting's heuristic are 20.19 and 43.94 compared to 19.57 and 38.11 achieved by Min–Max, when small and medium size problem instances are considered, respectively.

Next we analyze the effectiveness of the two improvements to the original Min–Max heuristic. We can notice that the use of the chain heuristic approach has improved the basic heuristic's results in almost all cases. It has decreased the quality of results, for only 1 of 48 test sizes, and in this case only slightly from 9.02 to 9.05. The level of improvement is best evaluated when we observe the average distance (error) from the Beam Search which we consider as a lower bound for the problem. This error given in percent has been decreased from 13.20, 12.52 to 11.45, 10.61 for small and medium sized instances, respectively.

When the combination of both improvements are added to the Min–Max heuristic, the results are further improved to 10.68, 9.10 when the average error is observed. Note that although the addition of the checks for stacks becoming filled improves the average results it has also degraded the quality for some test instances, when compared to the chain heuristic. This negative effect is most noticeable in smaller problem cases, especially when there is a low number of tiers.

To better analyze the effect of the proposed improvements to the original Min–Max Heuristic we have also included Tables 2, 4 which represent the standard deviation and range for each of the problem sizes. More precisely we have calculated the improvement in percent corresponding to the number of reshuffle operations of the Chain and Chain F compared to the original method, for each of the 40 problem instances inside of one problem size. There is a slight discrepancy of the results in Tables 2, 4 and 1, 3; this is due to the fact that in the first pair we show the relative change for a normalized number of moves and in the second pair only the number of moves. From these results it is noticeable, that although the improvements did have an average improvement compared to the original method in some of the problem instances they degraded the quality of the solution. This fact is more noticeable in the case of the Chain F heuristic where both the level of improvement and degradation are more significant, but overall the positive effect is greater.

We have also compared the effectiveness of the two improvements for large problem instances having 11–12 stacks with the maximal number of tiers going up to 12. The results of tests for large problem instances are given in Table 5. In this table we compare only the improved heuristics to the one proposed by Caserta et al. since data for the other methods was not available. We first notice that the effectiveness of the improved heuristic

Table 4
The range and standard deviation of the level of improvement for individual problem instances for medium sized bays given in percent of the Chain and Chain F heuristics compared to the Min–Max heuristic.

Tiers × stacks	Chain			Chain F		
	Imp.	Range	Stdev.	Imp.	Range	Stdev.
3 × 7	−0.23	(−9, 0)	1.42	−1.63	(−18, 11)	5.69
4	−0.56	(−12, 7)	3.08	1.35	(−11, 10)	5.11
5	0.85	(−5, 8)	2.45	1.69	(−12, 15)	5.85
6	0.64	(−11, 11)	4.82	0.35	(−18, 14)	8.05
7	1.56	(−7, 15)	4.12	2.40	(−17, 23)	8.05
8	1.50	(−7, 14)	4.88	3.90	(−15, 16)	7.04
3 × 8	0.00	(0, 0)	0.00	0.99	(−8, 9)	3.24
4	0.83	(0, 16)	2.92	0.83	(0, 16)	2.92
5	1.06	(−6, 12)	3.19	2.61	(−7, 20)	6.90
6	1.94	(−12, 27)	5.54	3.62	(−6, 27)	6.67
7	1.60	(−5, 12)	4.14	2.03	(−7, 19)	7.09
8	1.68	(−9, 14)	4.20	2.37	(−17, 15)	6.74
3 × 9	0.19	(0, 8)	1.20	0.21	(−13, 9)	4.06
4	0.52	(0, 9)	1.95	−0.40	(−14, 10)	5.49
5	1.68	(−4, 18)	4.12	0.77	(−21, 20)	8.08
6	0.72	(−6, 10)	3.11	3.32	(−16, 19)	6.80
7	3.08	(−9, 17)	5.12	4.04	(−6, 13)	5.29
8	1.50	(−12, 9)	3.99	3.40	(−7, 14)	5.96
3 × 10	0.00	(0, 0)	0.00	0.23	(−10, 17)	4.31
4	1.06	(−11, 7)	2.98	1.32	(−9, 14)	4.97
5	1.89	(−5, 11)	3.62	2.99	(−8, 14)	4.53
6	1.98	(−4, 14)	3.78	2.49	(−7, 14)	4.95
7	1.97	(−5, 10)	3.41	2.90	(−9, 23)	6.51
8	2.03	(−13, 11)	5.03	4.96	(−9, 17)	7.36

The higher level of improvement for the two heuristics has been underlined.

grows with the increasing number of stacks and tiers. When only the chain heuristic approach is used, the improvement varies from less than 1% for problems with a low number of tiers, up to almost 4% for a higher number of tiers. In the tests performed with large problem instances the combination of improvements has proven to be very effective. Compared to the chain heuristic it has degraded the quality of solutions in only one case. It has achieved a maximal improvement of even more than 8%, and around 5% on average when compared to Caserta et al.'s heuristic.

5. Conclusion

In this paper we have presented a new heuristic approach for solving the BRP. In a vast majority of greedy algorithms, the heuristic is based only on the block that is being relocated and the current state of the bay. We introduced a new heuristic approach in which when deciding where to relocate a single block we also take into account the properties of the block that will be moved next. This idea is illustrated by improving the Min–Max heuristic for the BRP. We also presented a simple improvement for the Min–Max heuristic by taking into account if a stack will reach a maximal tier if a block is moved to it.

We compared the new heuristic to several existing methods of this type on test data of a wide range of sizes. In our tests we have shown that when our two improvements are added to the Min–Max heuristic, the new method achieved the best results in almost all of the tested cases. It is important to point out that the level of improvement has grown with the size of the container bay. The improvement is significant, when compared to the basic Min–Max and has been on average around 5%, reaching even more than 8%, when large problem instances are considered. It is important to point out that for getting this level of improvement, there was almost no extra calculation time.

The proposed chain heuristic has clearly shown its advantages to the basic method. We believe that this concept can also be successfully used in combination with other heuristics for the BRP. In the future we plan to develop a more general formulation of the chain heuristic that could take into account a higher

Table 5
Comparison of different heuristics for BRP on large problem cases. Caserta et al. is our implementation of the Min–Max algorithm. Chain, Chain F are the Min–Max heuristic combined with only the chain heuristic, or with both improvements.

Tiers × stacks	Caserta et al.	Chain		Chain F	
		Average	Improvement	Average	Improvement
3 × 11	14.37	14.35	0.14	14.30	0.49
4	24.50	24.47	0.12	24.27	0.94
5	37.95	37.52	1.13	37.55	1.05
6	52.60	51.85	1.43	51.17	2.72
7	73.15	71.75	1.91	69.45	5.06
8	93.80	92.47	1.41	90.42	3.60
9	118.07	115.50	2.18	111.85	5.27
10	144.25	139.90	3.02	134.07	7.06
11	167.22	164.77	1.47	158.05	5.48
12	198.10	196.40	0.86	187.20	5.50
3 × 12	15.05	15.07	−0.13	15.05	0.00
4	28.12	27.80	1.14	27.52	2.13
5	42.02	42.00	0.05	41.52	1.19
6	58.35	57.15	2.06	56.37	3.39
7	81.25	79.35	2.34	76.90	5.35
8	102.52	99.12	3.32	96.95	5.43
9	125.57	124.10	1.17	120.72	3.86
10	150.05	145.90	2.77	142.55	5.00
11	183.97	180.60	1.83	174.97	4.89
12	220.80	212.32	3.84	202.70	8.20

The values of the best average solutions have been underlined.

number of following blocks without a significant increase in calculation time.

References

Caserta, M., Schwarze, S., & Voß, S. (2009). A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. In C. Cotta & P. Cowling (Eds.), *Evolutionary computation in combinatorial optimization. Lecture notes in computer science* (Vol. 5482, pp. 37–48). Berlin: Springer. ISBN 978-3-642-01008-8.

Caserta, M., Schwarze, S., & Voß, S. (2011a). Container rehandling at maritime container terminals. In J. W. Böse (Ed.), *Handbook of terminal planning. Operations research/computer science interfaces series* (Vol. 49, pp. 247–269). New York: Springer. ISBN 978-1-4419-8407-4.

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. ISSN 0377-2217.

Caserta, M., Voß, S., & Sniedovich, M. (2011b). Applying the corridor method to a blocks relocation problem. *OR Spectrum, 33*, 915–929. ISSN 0171-6468.

Duinkerken, M., Dekker, R., Kurstjens, S., Ottjes, J., & Dellaert, N. (2006). Comparing transportation systems for inter-terminal transport at the Maasvlakte container terminals. *OR Spectrum, 28*, 469–493. ISSN 0171-6468.

Duin, C., & Voß, S. (1999). The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks, 34*, 181–191.

Hussein, M. & Petering, M., (2012). 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, 1–8, 2012.*

Kim, K. H., & Hong, G.-P. (2006). A heuristic rule for relocating blocks. *Computers & Operations Research, 33*(4), 940–954. ISSN 0305-0548, doi:10.1016/j.cor.2004.08.005.

Lee, Y., & Lee, Y.-J. (2010). A heuristic for retrieving containers from a yard. *Computers & Operations Research, 37*(6), 1139–1147. ISSN 0305-0548.

Murty, K., Wan, Y.-W., Liu, J., Tseng, M. M., Leung, E., Lai, K.-K., et al. (2005). Hongkong International Terminals Gains Elastic Capacity Using a Data-Intensive Decision-Support System. *Interfaces, 35*(1), 61–75.

Petering, M. E. H., & Hussein, M. I. J. (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.

Stahlbock, R., & Voß, S. (2008). Operations research at container terminals: A literature update. *OR Spectrum, 30*, 1–52.

Ünlüyurt, T., & Aydin, C. (2012). Improved rehandling strategies for the container retrieval process. *Journal of Advanced Transportation, 46*(4), 378–393. ISSN 2042-3195.

Wan, Y., Liu, J., & Tsai, P. (2009). The assignment of storage locations to containers for a container stack. *Naval Research Logistics, 56*(8), 699–713. ISSN 0305-0548.

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.*

Zhang, C. (2000). Resource planning in container storage yard, PhD Dissertation, Department of Industrial Engineering, Ph.D. thesis, Department of Industrial Engineering, The Hong Kong University of Science and Technology.