THE CORRIDOR METHOD – A GENERAL SOLUTION CONCEPT WITH APPLICATION TO THE BLOCKS RELOCATION PROBLEM

Marco Caserta [(a)], Stefan Voß[(b)], Moshe Sniedovich[(c)]

[(a)] Institut für Wirtschaftsinformatik, Universität Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
[(b)] Institut für Wirtschaftsinformatik, Universität Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
[(c)] Department of Mathematics and Statistics, The University of Melbourne, Australia

[(a)] marco.caserta@uni-hamburg.de, [(b)] stefan.voss@uni-hamburg.de, [(c)] m.sniedovich@ms.unimelb.edu.au

ABSTRACT
The Corridor Method is a solution concept which may be characterized as a method-based metaheuristic. That is, based on a given algorithm which is meant to work well at least on small sized instances of a specific type of optimization problem, it defines one or more neighborhoods which seem to be well suited for the specific problem and the given algorithm. These neighborhoods may be viewed as corridors around given solutions. Experiments show that this type of approach is a successful hybridization between exact and metaheuristic methods. We show successful applications for the block relocation problem arising, e.g., at container terminals.

Keywords: corridor method, blocks relocation problem, container terminal

1. INTRODUCTION

In this paper we present a dynamic programming inspired metaheuristic called corridor method along with its application upon the blocks relocation problem in block stacking systems, as found, e.g., in the stacking of container terminals in a yard. It can be classified as a local search based metaheuristic in that the neighborhoods that it deploys are method-based. By this we mean that the search for a new candidate solution is carried out by a fully-fledged optimization method which generates an optimal solution over the neighborhood. The neighborhoods are thus constructed to be suitable domains for this optimization method. Typically these neighborhoods are obtained by the imposition of exogenous constraints on the decision space of the target problem and, therefore, must be compatible with the method used to search these neighborhoods. This is in sharp contrast to most traditional metaheuristics where neighborhoods are move-based, i.e., they are generated by subjecting the candidate solution to small changes called moves.

While conceptually the method-based paradigm applies to any optimization method, in practice it is best suited to support optimization methods, such as dynamic programming, where it is easy to control the size of a problem, hence the complexity of algorithms, by means of exogenous constraints. The essential features of the method may be illustrated in the context of well-known combinatorial optimization problems where exponentially large dynamic programming inspired neighborhoods are searched by a linear time/space dynamic programming algorithm.

This paper has the following structure: Section 2 presents the blocks relocation problem along with a dynamic programming based formulation; Section 3 illustrates how the Corridor Method can be implemented to effectively tackle the problem at hand; Section 4 offers computational results and, finally, Section 5 concludes with some remarks.

2. THE BLOCKS RELOCATION PROBLEM

Increasing containerization and competition among seaport container terminals have become quite remarkable in worldwide international trade. Operations are nowadays unthinkable without effective and efficient IT use and appropriate optimization (management science and operations research) methods. Besides enabling efficient data interchange between supply chain partners, related information systems need to support terminal operators, shipping companies and even port authorities.

In container terminals, it is common practice to store outbound containers in the yard before loading them into a vessel. To be able to face competition among terminals and to guarantee a high level of service, operators must reduce unproductive time at the port; see, e.g., Stahlbock and Voß (2008) and the references given there for a comprehensive survey.

Relocation is one of the most important factors contributing to the productivity of operations at storage yards or warehouses (Yang and Kim, 2006). A common practice aimed at effectively using limited storage space is to stack blocks along the vertical direction, whether they be marine containers, pallets, boxes, or steel plates (Kim and Hong, 2006). Given a heap of blocks, relocation occurs every time a block in a lower tier must be retrieved before blocks placed above it. Since blocks in a stack can only be retrieved following a LIFO (Last In First Out) discipline, in order to retrieve the low-tier block, relocation of all blocks on top of it will be

necessary. Figure 1 illustrates how the block stacking technique is used at a bay. Each vertical heap is called stack. A stack is made up by a number of tiers, which define the height of the stack. A bay is the collection of stacks and the width of the bay is given by the number of stacks. In Figure 1 the order in which a block is to be retrieved is indicated by a progressive number. Consequently, in order to pickup block 1, blocks 5 and 4, in this order, must first be relocated to either stack 1 or 3.

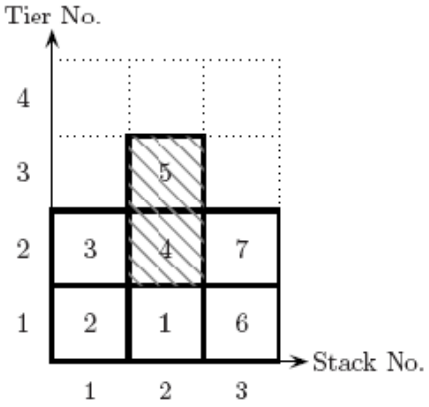

Figure 1: An example of a bay with m = 3 stacks and n = 7 blocks.

If the current configuration of the bay is taken as fixed, one might be interested in finding the sequence of moves that should be executed, while retrieving blocks according to a given sequence, in order to minimize the overall number of relocation moves. While in the shuffling problem containers are rearranged but not removed, in the on-line version of the problem, at each step, a container is removed from the bay, hence reducing the number of containers in the bay until all containers have been picked up from the bay. Exact as well as approximate algorithms have been proposed to minimize the number of relocations while retrieving blocks. For example, Watanable (1991) proposed the use of an accessibility index to forecast the number of relocation movements. With a similar approach, Castillo and Daganzo (1993) and Kim (1997) proposed heuristic rules to estimate the number of relocations for inbound containers. A mathematical formulation and a branch and bound approach, along with an effective heuristic scheme for the blocks relocation problem are presented in Kim et al. (2000) and in Kim and Hong (2006).

In this paper, in a fashion similar to what is presented in Kim and Hong (2006), we consider the initial bay configuration as fixed and the sequential order of pickup as known in advance. Given this fixed initial arrangement, we are interested in finding the relocation pattern at each pickup operation in such a way that the total number of relocation moves within a bay is minimized.

The general idea of this paper is based upon the introduction of a dynamic programming scheme that captures all the possible states of the bay at any given time. Given an incumbent bay configuration and a target block to retrieve, we distinguish between two cases: (i) the block to be retrieved is *free*, which is, no block is currently above it. In this case, the only acceptable decision is to retrieve the target block and place it into its final destination. On the other hand, (ii) if at least one block is currently placed upon the target block, we enumerate all the possible relocation strategies for the uppermost block and move this block to another stack, giving rise to a new bay configuration. This enumeration process is repeated until case (i) is reached, after which the current target block is retrieved and the next target block is addressed. The process terminates when the last block to be retrieved is freed up.

Let us consider the case of a bay with N blocks, indicated with $\{1, \dots, N\}$, in which the first n blocks must be retrieved, with $n \leq N$. Without loss of generality, we assume that not necessarily all the blocks must be retrieved. Let us indicate with $i \in \{1, \dots, m\}$ a stack in the bay and with $k \leq n$ the current block to be retrieved. Throughout this section, we will use the bay of Figure 1 as reference example.

We now introduce the following assumptions:

H1: As in Kim and Hong (2006), pickup precedences among blocks are known in advance. We indicate the pickup precedence with a number, where blocks with lower numbers have higher precedence than blocks with higher numbers (e.g., in Figure 1, the pickup precedences are $1 < 2 < \cdots < 7$).

H2: When retrieving a target block, we are allowed to relocate only blocks found above the target block in the same stack using a LIFO policy (e.g., in Figure 1, when picking up block 1, we are forced to relocate blocks 5 and 4, in this exact order).

H3: Relocation is allowed only to other stacks within the same bay (e.g., in Figure 1, when relocating blocks 5 and 4, they can only be moved to either stack 1 or 3).

H4: Relocated blocks can be put only on top of other stacks, i.e., no rearrangement of blocks within a stack is allowed (e.g., in Figure 1, when relocating blocks 5 and 4, these can only be placed on top of blocks 3 and 7).

In the following, let us define the basic elements of the dynamic programming (DP) model:

State variable: Let us indicate with $s = (k, i, t, C)$ the state variable, where $k \in \{1, \dots, n\}$ is the block to be retrieved, $i \in \{1, \dots, m\}$ is the stack in which the target block is found, t is the list of blocks above the target block and C is the configuration of the remaining blocks (e.g., with respect to Figure 1, we have $k = 1, i = 2, t = \{5, 4\}$, and $C = \{\{3, 2\}, \{7, 6\}\}$).

Decision variable: At each step, one of two different cases arises, namely *(i)* the target block has no other blocks placed above and can currently be retrieved and placed outside of the bay, i.e., $t = \emptyset$. In this case, the only alternative is to retrieve the target block and to

place it into its final destination; *(ii)* the target block cannot be retrieved since at least one block is still above it, i.e., $t \neq \emptyset$. Let us indicate with τ the uppermost block in the sequence t, which is, the block that is currently on top in stack i. In this case, the decision is about identifying which stack block τ should be relocated to. Let us indicate with x such a stack and with $D(s)$ the set of all feasible values of x with respect to the current state s (e.g., in Figure 1, we have $\tau = 5$ and $D(s) = \{1, 3\}$, that is, the next decision concerns where to relocate block 5 and the only feasible moves are either to move it to stack 1 or to stack 3).

State transition function: Let us indicate with $s' = (k', i', t', C')$ the state obtained by applying decision $x \in D(s)$ to the current state s, which is, $s' = T(s, x)$. Here T represents the state transition function. As previously mentioned, two different cases may arise: *(i)* $t = \emptyset$; or *(ii)* $t \neq \emptyset$. In case *(i)*, we have that $k' = k + 1$, $i'$is the stack in which block $k + 1$ is currently located, $t'$is a new list of blocks currently above the target block $k + 1$ and, finally, $C' = C$ is the configuration of the remaining blocks. On the other hand, in case *(ii)*, it is easy to see that $k' = k, i' = i, t' = t \setminus \{\tau\}$, and C'depends on the application of move x to block τ (e.g., in Figure 1, let us suppose that $x = 1$. In this case, the new state is $s' = T(s, 1) = (1, 2, \{4\}, C')$, where $C' = \{\{5, 3, 2\}, \{7, 6\}\}$).

Functional equation: The DP "backward" functional equation is

$$f(s) = 1 + \min_{x \in D(s)}\{f(T(s, x)\} \quad (1)$$

where $s = (k, i, t, C)$ indicates the current state, and $T(s, x)$ is the state transition function that accounts for the application of decision x upon the current state s. We set $f(s) = 1$, for $s = (n, i, \emptyset, C)$, which is the cost of retrieving a block from the bay and moving it to its final destination. More formally, we can explicitly distinguish between the aforementioned cases *(i)* and *(ii)*. Consequently, the functional equation can be rewritten, for $k = n - 1, \dots, 1$, as

$$f(k, i, t, C) = \begin{cases} 1 + f(k + 1, i', t', C), & t = \emptyset \\ 1 + \min\limits_{x \in D(s)} \{f(k, i, t \setminus \{\tau\}, C')\}, & t \neq \emptyset \end{cases} \quad (2)$$

with $f(n, i, \emptyset, C) = 1$.

It is easy to see that one of the major drawbacks of the proposed dynamic programming scheme lies in the exponentially large number of entries in the dynamic programming evaluation table. Consequently, the size of the evaluation table can become very large after only a few steps of the dynamic programming algorithm. In the next section we illustrate how this major obstacle can be overcome.

3. THE CORRIDOR METHOD FOR THE BLOCKS RELOCATION PROBLEM

The Corridor Method (CM) has been presented by Sniedovich and Voß (2006) as a hybrid metaheuristic, linking together mathematical programming techniques with heuristic schemes. The basic idea of the CM relies on the use of an exact method over restricted portions of the solution space of a given problem. Given an optimization problem *P*, the basic ingredients of the method are a very large feasible space *X* and an exact method *M* that could easily solve problem *P* if the feasible space were not too large. However, since, in order to be of interest, in general the size of the solution space grows exponentially with respect to the input size, the direct application of method *M* to solve *P* becomes unpractical when dealing with real-world instances, which is, when *X* becomes larger.

Let us now consider how the CM can be applied to the blocks relocation problem. As mentioned in Section 2, the main drawback of the proposed method is the exponential growth of the number of reachable states. We define a "two-dimensional" corridor around the current configuration, in such a way that the number of states generated from the current configuration is limited. Given a current bay configuration s, the number of new configurations that can be generated starting from s is equal to $|D(s)|$. Consequently, in order to reduce the number of generated states, one can apply exogenous constraints that impose horizontal as well as vertical limits upon the bay. For example, horizontal limits could be introduced by reducing the number of stacks to which blocks can be relocated, as well as vertical limits, by establishing a maximum height, in terms of number of blocks in the same stack.

Let us now formally define the "constrained" neighborhood induced by the application of the CM upon a given configuration. Let us indicate with $s = (k, i, t, C)$ the current bay configuration, where $C = \{c_1, \dots, c_m\} \setminus \{c_i\}$ indicates all the stacks of the bay excluding stack i. Let us indicate with $|c_i|$ the number of blocks currently on stack i. Given two parameters δ and λ, we define the set of restricted admissible decisions as

$$D(s, \delta, \lambda) = \{x \in \{1, \dots, m\} \setminus \{i\} : i - \delta \leq x \leq i + \delta, |c_x| \leq \lambda\}. \quad (3)$$

Finally, we define the restricted neighborhood of the current configuration s, i.e., the set of "feasible" bay configurations that can be created from s as

$$N(s) = \{s' : s' = T(S, x), x \in D(s, \delta, \lambda)\}. \quad (4)$$

Consequently, the size of the neighborhood can be made arbitrarily small by changing the values of δ and λ. For this reason, we can say that the "corridor" around the incumbent bay configuration is defined by imposing *exogenous constraints* on the solution space of the problem via calibration of parameters δ and λ.

4. COMPUTATIONAL RESULTS

In this section we present computational results on randomly generated instances. All tests presented in this section have been carried out on a Pentium IV Linux Workstation with 512Mb of RAM. The algorithm has been coded in C++ and compiled with the GNU C++ compiler using the -O option.

We designed an experiment that resembles that of Kim and Hong (2006). We divided our computational tests in two parts: *(i)* tests on small-medium size instances, for which an exact solution can be computed using the dynamic programming scheme. For these instances, we present a comparison of our algorithm with respect to the optimal solution as well as with respect to the solution found by the algorithm of Kim and Hong (2006), running the code provided by the authors on our randomly generated instances; and *(ii)* tests on large scale instances, for which the optimal solution is unknown. In order to measure the solution quality of our algorithm, we compare our results with those obtained running the algorithm of Kim and Hong (2006) on the same set of instances.

The random generation process takes as input two parameters, the number of stacks m and the number of tiers h, and randomly generates a rectangular bay configuration of size $n = h \times m$, where n indicates the total number of blocks in the bay. For each combination of m and h we generated 40 different instances.

In Table 1 we compare the results of the proposed scheme with those obtained running the code of Kim and Hong (2006) on the same set of instances. It is worth noting that all values reported in the table are *average* values, computed over 40 different instances of the same class. This helps in offsetting instance specific biases in the reported results. In addition, we fixed a maximum computational time for the CM of 60 seconds, after which the algorithm was stopped. Clearly, since the CM is based on a dynamic programming scheme, whenever the algorithm is truncated, no solution is returned.

In Table 1 and Table 2, the first and second columns define the size of problem instances, in terms of number of tiers and number of stacks. The third and fourth columns report average number of moves and computational time of the algorithm of Kim and Hong (2006) (called KH for short). Columns five and six report the same information, average number of moves and average computational time, for the proposed CM. Column seven reports the gap between the solution of the CM and the optimal solution obtained by using the dynamic programming scheme (omitted in Table 2 because instances were too large to be solved via dynamic programming). The gap is computed as:

$$\gamma = \frac{z^{CM} - z^*}{z^*} \qquad (5)$$

where z^* is the optimal solution found by the dynamic programming scheme and z^{CM} is the average best solution found by the proposed CM scheme. Finally, the last two columns provide a measure of the corridor, in terms of width (δ) and height (λ). We solved each instance with different combinations of $\delta \in \{1, 2, \dots, m/2\}$ and $\lambda \in \{h+1, h+2, \dots, 1.5h\}$. We report the values used to obtain the best solution in the shortest computational time. It is worth noting that, for the sake of consistency with the results reported in Kim and Hong (2006), all results reported in this section with respect to the number of moves (No.) only count the number of relocations and do not take into account the final retrieval of the target blocks to their final destinations.

In addition, in order to further reduce the stochastic effects of the algorithm, we run the algorithm with the same set of parameters δ and λ five times on the same instance. Consequently, given an instance class $h \times m$, we solved each one of the 40 instances $5 \times |\Lambda| \times |\Delta|$ times. In the table we report the average values over all the runs of a given instance class.

Table 1: Computational Results on Small Size Instances.

Bay Size		KH		CM			Corridor	
h	m	No.	Time†	No.	Time†	γ	δ	λ
3	3	7.1	0.1	5.4	0.10	0.00	1	4
3	4	10.7	0.1	6.5	0.10	0.00	1	4
3	5	14.5	0.1	7.3	0.10	0.00	1	4
3	6	18.1	0.1	7.9	0.15	0.00	2	4
3	7	20.1	0.1	8.6	0.10	0.01	2	4
3	8	26.0	0.1	10.5	0.20	0.01	2	4
4	4	16.0	0.1	9.9	0.20	0.02	2	5
4	5	23.4	0.1	16.5	0.50	0.01	2	5
4	6	26.2	0.1	19.8	0.50	0.03	2	5
4	7	32.2	0.1	21.5	0.50	0.03	2	5

† : CPU seconds on a Pentium-IV 512Mb RAM.

In Table 1, the "smaller" instances in the upper part of the table, i.e., 3×3 to 3×6, are solved to optimality by the CM. Consequently, a first observation of our results is related to the effectiveness of the CM in solving small instances to optimality in a very short computational time. For all the other instances, the algorithm compares favorably with the Kim and Hong (2006) algorithm, especially when dealing with large scale instances. In order to provide a further indication of the robustness of the algorithm, in Figure 2 we graphically present the variability of the results on the largest instances. As shown in Figure 2, the algorithm is quite robust with respect to the parameter values δ and λ as well as with respect to the initial configuration of the bay.

Table 2: Computational Results on Medium Size Instances.

Bay Size		KH		CM		Corridor	
h	m	No.	Time†	No.	Time†	δ	λ
5	4	23.7	0.1	16.6	0.5	2	6
5	5	37.5	0.1	18.8	0.8	2	6
5	6	45.5	0.1	22.1	0.8	2	6
5	7	52.3	0.1	25.8	1.43	1	7
5	8	61.8	0.1	30.1	1.46	1	6
5	9	72.4	0.1	33.1	1.41	1	6
5	10	80.9	0.1	36.4	1.87	1	6

† : CPU seconds on a Pentium-IV 512Mb RAM.

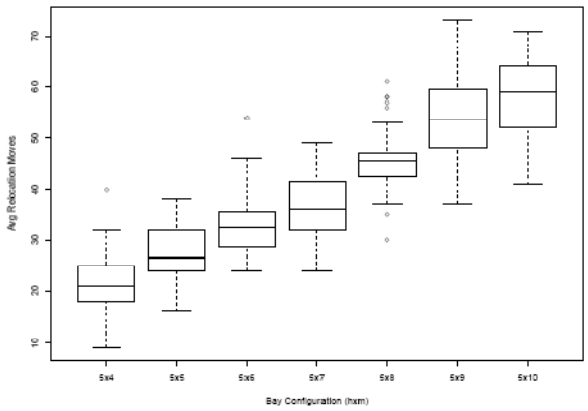

Figure 2: Variability results.

5. CONCLUSION

In this paper we have presented a metaheuristic-based algorithm for the Blocks Relocation Problem, in which one is given a sequence of blocks to be retrieved from a bay according to a fixed set of precedences. The objective is to find the blocks relocation pattern that minimizes the total number of movements required to comply with the retrieving sequence. This problem finds applications in a wide spectrum of real-world situations, where stacking techniques are used to reduce space usage, e.g., at a container terminal yard. We have first proposed a dynamic programming algorithm that can be used to find the optimal solution to the problem. However, since the size of the search space grows exponentially with respect to the input size, the dynamic programming approach cannot be used in real time when dealing with medium and large scale instances. For this reason, we tackled the problem by designing a Corridor Method inspired algorithm, in which a two-dimensional "corridor" is build around the incumbent yard configuration. The imposition of exogenous constraints on the target problem sensibly reduces the size of the solution space, making the use of the "constrained" dynamic programming scheme practical even on very large instances.

REFERENCES

Castilho, B. and Daganzo, C. (1993). Handling strategies for import containers at marine terminals. *Transportation Research B*, 27(2):151–66.

Kim, K. H. (1997). Evaluation of the number of rehandles in container yards. *Computers & Industrial Engineering*, 32(4):701–711.

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

Kim, K. H., Park, Y. M., and Ryu, K. R. (2000). Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124:89–101.

Sniedovich, M., Voß, S., 2006. The corridor method: A dynamic programming inspired metaheuristic. *Control and Cybernetics* 35, 551 - 578.

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

Watanabe, I. (1991). Characteristics and analysis method of efficiencies of container terminal: an approach to the optimal loading/unloading method. Container Age, 3:36–47.

Yang, J. H. and Kim, K. H. (2006). A grouped storage method for minimizing relocations in block stacking systems. *Journal of Intelligent Manufacturing*, 17:453–463.