COLUMN GENERATION FOR THE CONTAINER RELOCATION PROBLEM

Elisabeth Zehendner
Ecole des Mines de Saint-Etienne, CMP Georges Charpak
13541 Gardanne, France
zehendner@emse.fr

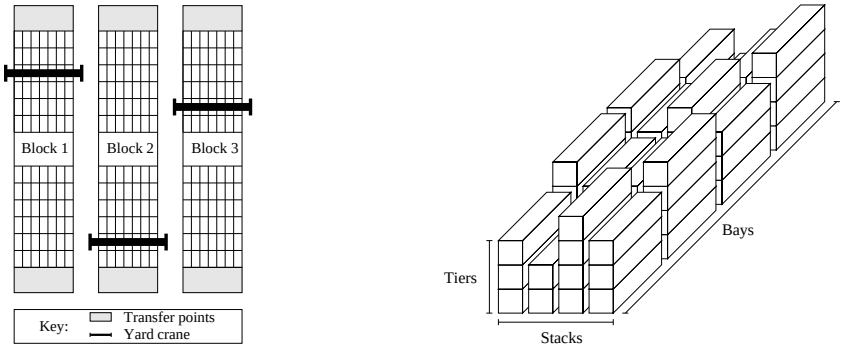
Dominique Feillet
Ecole des Mines de Saint-Etienne, CMP Georges Charpak
13541 Gardanne, France
feillet@emse.fr

Abstract

Container terminals offer transfer facilities to move containers from vessels to trucks, trains and barges and vice versa. Within the terminal the container yard serves as a temporary buffer where incoming containers are piled up in stacks. Only the topmost container of each stack can be accessed. If another container has to be retrieved, containers stored above it must be relocated first. Containers need to be transported to a ship or to trucks in a predefined sequence as fast as possible. Generally, this sequence does not match the stacking order within the yard. Therefore, a sequence of retrieval and relocation movements has to be determined that retrieves containers from the bay in the prescribed order with a minimum number of relocations. This problem is known as the container relocation problem. We apply an exact and a heuristic column generation approach to this problem. First results are very promising since both approaches provide very tight lower bounds on the minimum number of relocations.

1 Introduction

Container terminals serve as exchange hubs in intermodal transportation. They offer transfer facilities to move containers from vessels to trucks, trains and barges and vice versa. Container terminals consist of the quayside, the landside and the yard. Vessels are loaded and unloaded by quay cranes at the quay. The landside is the terminal's interface with the inland transportation system (rail, road, waterway). The yard serves as a temporary buffer for containers. Our study deals with grounded storage operations where containers are



(a) A container yard (b) A single block

Figure 1: Illustration of blocks, bays, stacks and tiers

stacked on the ground in several levels. The yard is divided into several blocks and each block consists of several bays. A bay consists of 6 to 10 stacks and each stack of 4 to 7 tiers. Figure 1 illustrates these terms. Since containers are piled up, only the topmost container of each stack can be accessed. If another container needs to be moved, containers that are placed on top of the required one have to be relocated. These unproductive movements are called reshuffles or rehandles.

The objective of yard optimization is to minimize storage and retrieval times of containers while maximizing storage space utilization. To achieve this goal several problems have to be addressed. Inbound and outbound containers have to be allocated to storage blocks while balancing the workload between blocks. Then, each container has to be assigned to a slot within the block. The objective is to avoid reshuffles at a later time. However, few information about the retrieval time of a container is known at the moment it has to be stored and reshuffling later on cannot be avoided. Container terminals use less busy periods to rearrange the yard as new information becomes available. The aim is to reduce the number of reshuffles necessary during the subsequent retrieval process. This rearrangement cannot always be done. If the yard is not prearranged the objective is to retrieve containers from the bay in the given sequence as fast as possible. This paper deals with this last problem which is known as the container (or block) relocation problem. Section 2 introduces the container relocation problem and discusses existing work. Section 3 details our exact and heuristic column generation approaches and the underlying bounding mechanisms to solve this problem. Section 4 compares the results of our column generation approaches with the results of a binary model and a simple heuristic. Section 5 indicates the continuation of this work.

2 Container relocation problem

Vessels have to be loaded according to a stowage plan. This stowage plan imposes the loading sequence of containers and thereby also the retrieval sequence of containers from

the yard. Trucks are also loaded in a specified order (e.g., FIFO). The retrieval sequence may either specify precedence constraints among single containers or among groups of containers. In order to minimize vessel and truck turnaround times retrieval operations have to be executed as fast as possible. It is impossible to stack containers in a non-blocking way since only partial information about future retrievals is available at the time the container is stored. Therefore, non-productive reshuffles are necessary during the retrieval process. A common procedure is to relocate containers within the bay since relocations between bays are very time consuming. The time it takes to retrieve a container from the topmost position of a stack is similar for all stacks. The number of reshuffles has thus the biggest influence on the retrieval time. The overall retrieval time is shortest for a minimum number of relocations.

This motivates the container relocation problem. Its objective is to determine a sequence of movements with a minimum number of reshuffles to retrieve containers from the bay in the given order. The problem definition relies on assumptions A1 to A6. Most studies also include assumptions A7.

A1: The initial bay layout and the retrieval order of single containers or groups of containers are known in advance.

A2: No new containers arrive during the retrieval process.

A3: Only the topmost container of a stack can be picked up. A relocated container can only be put on the top of another stack or on the ground.

A4: The maximum number of stacks and tiers per bay are given and containers are only relocated within the bay.

A5: Containers in the same bay can be piled up in any order.

A6: The distance traveled within one bay has no impacts on the time to reshuffle or to retrieve a container.

A7: Only containers located in the same stack as and above the current target container are allowed to be relocated.

We give an overview of studies dealing with the container relocation problem. For a broader review of literature on container stacking refer to Caserta et al. [2]. The container relocation problem is shown to be NP-hard by Caserta et al. [3]. This implies that only small instances can be solved with exact methods in reasonable time. That is why only few papers present exact solution approaches. Lee and Hsu [8] present different integer linear programming models based on a multi-commodity flow problem with a set of side constraints for several stacking problems. An adaptation to the container relocation problem is also shown. Caserta et al. [3] present two binary programming models. The models

contain state variables to represent the bay layout and movement variables to represent retrievals and relocations of containers. They also propose a simple heuristic based on the computation of a stack score.

Kim and Hong [7] study the problem with priorities among single containers and among groups of containers. They use a branch and bound algorithm that branches over all possible bay layouts resulting from the retrieval of a target container. They also propose a heuristic rule based on the expected number of additional relocations based on probabilities for container relocations. In a similar approach, Wu and Ting [10] apply different branch and bound based heuristics using different simple relocation strategies. Zhang et al. [11] present an iterative deepening A* algorithm (IDA*) and introduce new lower bound measures. Forster and Bortfeldt [6] present a tree search procedure. In contrast to all other studies, all containers and not only the ones stacked above the target container can be relocated. Their approach is based on a classification of feasible moves, a greedy approach to find a initial solution, a lower bound for the number of moves and a branching scheme.

Wu et al. [9] use tabu-search to tackle the problem. They use a one-dimensional vector to represent reshuffle and retrieval operations. Two simple heuristics are used to generate the initial solution and two move operators are applied to obtain neighborhood solutions. Caserta et al. [5] present a dynamic programming scheme and use it in a metaheuristic fashion within the corridor method algorithm. The corridor method reduces the exponentially large number of possible states by adding exogenous constraints to limit the number of possible relocations. The dynamic programming scheme is then applied to the restricted solution space. Caserta et al. [1] and Caserta and Voß [4] present two implementations for a random based procedure. For a container to be relocated, a neighborhood of the current configuration that can be reached from the current bay layout is defined. A greedy score is calculated for each configuration and a roulette-wheel mechanism is applied to select the next configuration and its associated move. Caserta et al. [1] propose a binary description of the problem and apply this random based procedure. In Caserta and Voß [4] the corridor method is used to define the neighborhood of the current configuration within the random based procedure.

3 Column generation for the container relocation problem

Branch and bound techniques are often used to solve integer programming models. In these cases, the gap between the bound computed at each node of the search tree and the optimal integer solution should be minimized to speed up the solution process. Generally, linear relaxations of integer programming models provide weak bounds. Column generation may be used to tighten the linear relaxation and to improve the quality of the bounds.

Column generation calculates the linear relaxation of a reformulation of the original problem to determine a linear relaxation of the original problem. The relaxed reformulation is called master problem MP(Ω) where Ω is a large set of variables (also called columns). Generally, the optimal solution of MP(Ω) cannot be determined with a standard

linear programming approach due to the size of Ω. In the optimal solution of MP(Ω) most of the variables equal zero. The idea behind column generation is to use only a subset of variables to determine the optimal solution of MP(Ω). The so-called restricted master problem MP(Ω_r) is instantiated with a subset Ω_r of variables ($\Omega_r \subset \Omega$). Then, Ω_r is extended progressively, by the so-called subproblem, until an optimality criterion assures that the optimal solution of MP(Ω_r) is also the optimal solution of MP(Ω).

Column generation is an iterative approach where MP(Ω_r) and the subproblem are solved repeatedly. The subproblem determines variables in $\Omega \backslash \Omega_r$ that have the potential to improve the objective value of MP(Ω_r) (so-called variables with negative reduced costs). These calculations are based on the values of dual variables of the current solution of MP(Ω_r). Then, MP(Ω_r) is updated and resolved and the subproblem determines further variables $\Omega \backslash \Omega_r$ to be added to MP(Ω_r) for the new values of its dual variables. The process stops if no more variables in $\Omega \backslash \Omega_r$ have the potential to improve the objective value of MP(Ω_r).

3.1 Master problem

The master problem is a reformulation of the binary model presented in Caserta et al. [3]. They consider a bay with W stacks and H tiers. Each slot within the bay is addressed with coordinates (i, j) where $i \in \{1, \ldots, W\}$ and $j \in \{1, \ldots, H\}$. The initial configuration contains N containers, labeled $1, \ldots, N$. Containers have to be retrieved in ascending order, e.g. container 1 is the first one to be retrieved and container N the last one. At each time period t ($t \in \{1, \ldots, T\}$) exactly one container ($n = t$) is retrieved and several containers may be relocated. But, only containers blocking the container to be retrieved may be relocated. Their objective is to minimize the number of relocations.

They identify two sets of binary variables. Configuration variables b_{ijnt} represent the layout of the bay. All b_{ijn1} with $t = 1$ are parameters which indicate the initial configuration of the bay. Movement variables x_{ijklnt} and y_{ijnt} represent the relocation and retrieval of containers:

$$b_{ijnt} = \begin{cases} 1 & \text{if block } n \text{ is in } (i, j) \text{ at the beginning of period } t, \\ 0 & \text{otherwise;} \end{cases}$$

$$x_{ijklnt} = \begin{cases} 1 & \text{if block } n \text{ is relocated from } (i, j) \text{ to } (k, l) \text{ in period } t, \\ 0 & \text{otherwise;} \end{cases}$$

$$y_{ijnt} = \begin{cases} 1 & \text{if block } n \text{ is retrieved from } (i, j) \text{ in period } t, \\ 0 & \text{otherwise.} \end{cases}$$

To model the master problem for column generation we also use the configuration variables b_{ijnt}. We introduce new variables representing sequences of movements to retrieve and relocate containers. Each sequence determines a series of movements to retrieve one

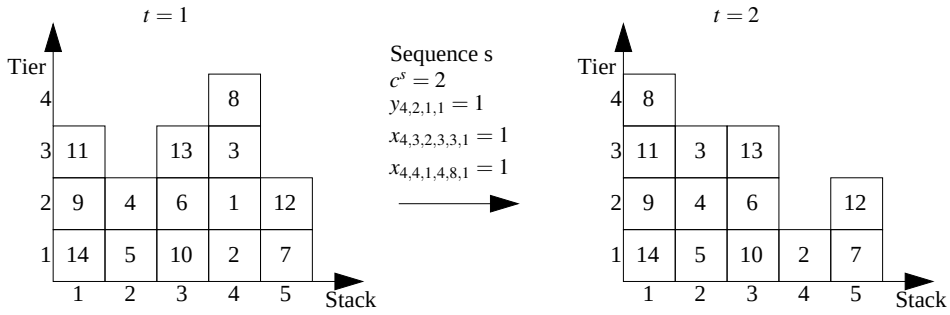


Figure 2: Transformation of the bay by applying a sequence

container or to retrieve one container and to relocate containers stacked above it. Let s design such a sequence. In this case, y^s_{ijnt} and x^s_{ijklnt} are parameters of the sequence with values fixed to 0 or 1. The cost of a sequence c^s is equal to the number of relocations to be executed if sequence s is applied ($c^s = \sum_i \sum_j \sum_k \sum_l \sum_n \sum_t x^s_{ijklnt}$). Figure 2 illustrates the transformation of the bay for a given sequence (only y^s_{ijnt} and $x^s_{ijklnt} \neq 0$ are indicated).

Remember that only container $n = t$ can be retrieved at period t. Let Ω^k design the set of all sequences that may be applied at period k ($k \in \{1,\ldots,T-1\}$). For all sequences $s \in \Omega^k$, all parameters y^s_{ijnt} and x^s_{ijklnt} with $t \neq k$ are zero. To solve the container relocation problem, we have to chose those sequences that empty the bay in the prescribed order with a minimum number of relocations. We introduce variables θ^s with

$$\theta^s = \begin{cases} 1 & \text{if sequence } s \text{ is applied,} \\ 0 & \text{otherwise.} \end{cases}$$

The the master problem for the container relocation problem can then be formulated as follows:

$$\min \sum_{t=1}^{T-1} \sum_{s \in \Omega^t} c^s \cdot \theta^s$$

s.t.

$$\sum_{n=t}^{N} b_{ijnt} \leq 1 \quad \forall i = 1,\ldots,W, j = 1,\ldots,H, t = 1,\ldots,T \tag{1}$$

$$\sum_{n=t}^{N} b_{ijnt} \geq \sum_{n=t}^{N} b_{ij+1nt} \quad \forall i = 1,\ldots,W, j = 1,\ldots,H-1, t = 1,\ldots,T \tag{2}$$

$$b_{ijnt+1} = b_{ijnt} + \sum_{s \in \Omega^t} \sum_{k=1}^{W} \sum_{l=1}^{H} \theta^s \cdot x^s_{klijnt} - \sum_{s \in \Omega^t} \sum_{k=1}^{W} \sum_{l=1}^{H} \theta^s \cdot x^s_{ijklnt} \tag{3}$$
$$\forall i = 1,\ldots,W, j = 1,\ldots,H, n = t+1,\ldots,N, t = 1,\ldots,T-1$$

$$b_{ijtt} - \sum_{s \in \Omega^t} \theta^s \cdot y^s_{ijtt} = 0 \quad \forall i = 1,\ldots,W, j = 1,\ldots,H, t = 1,\ldots,T-1 \tag{4}$$

$$\sum_{s \in \Omega^t} \theta^s = 1 \quad \forall t = 1,\ldots,T-1 \tag{5}$$

$$\theta^s \geq 0 \quad \forall t = 1,\ldots,T-1, s \in \Omega^t \tag{6}$$

$$b_{ijnt} \geq 0 \quad \forall i = 1,\ldots,W, j = 1,\ldots,H, n = t,\ldots,N, t = 2,\ldots,T \tag{7}$$

The objective function minimizes the total number of relocations. The last period T is not included since the last remaining container has to be retrieved from and no relocations occur. Note that at the beginning of period t only containers $n \geq t$ are in the bay. Constraint (1) makes sure that each slot (i, j) is occupied by at most one container. Constraint (2) prevents gaps within stacks. If a container needs to be retrieved from position (i, j) containers at positions (i, j') with $j' > j$ must also be moved. Moreover, no container can be relocated to a suspended position. Constraint (3) links the bay layout at period t with the layout at period $t+1$ via the relocations to be carried out for the chosen sequences. Constraint (4) makes sure that container $n = t$ is retrieved from the bay in period t. Constraint (5) imposes that exactly one sequence is chosen per period. Constraints (6) and (7) represent the domain definitions of the linear variables. Remember that all y^s_{ijnt} and x^s_{ijklnt} are binary parameters that are defined for each sequence s.

3.2 Restricted master problem

The restricted master problem is also defined by Constraints (1) to (7). It is instantiated with a subset of variables Ω^t_r ($\Omega^t_r \subset \Omega^t$ for at least one $t \in \{1,\ldots,T-1\}$). To determine variables to be added to the restricted master problem, a feasible solution of the restricted master problem has to be known. We use the heuristic presented in Caserta et al. [3] to determine such a subset. The idea of their heuristic is based on some simple relocation rules. A stack score s_i is determined for each stack i in the bay. s_i equals the highest priority (lowest n) in the stack or $N+1$ if the stack is empty. Suppose that container t has to be relocated. If there is at least one stack with $s_i > t$ (the stack contains no containers with higher priorities than container t), relocate container t to the stack with the smallest s_i, since stacks with large s_i are valuable. If there is no stack with $s_i > t$, then choose the stack with the highest s_i as container t will cause an additional relocation anyway.

3.3 Subproblem

The objective of the subproblem is to determine new sequences in $\Omega \backslash \Omega^t_r$ to be added to Ω^t_r that reduce the objective value of the restricted master problem. In other words, sequences

with negative reduced costs have to be determined. Let γ_{ijnt} be the dual variables of Constraint (3), δ_{ijt} the dual variables of Constraint (4) and μ_t the dual variables of Constraint (5). We are looking for sequences s satisfying Constraint (8).

$$c^s - \sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{n=t+1}^{N} x^s_{klijnt}\cdot\left(\gamma_{ijnt}-\gamma_{klnt}\right) + \sum_{i=1}^{W}\sum_{j=1}^{H} y^s_{ijtt}\cdot\delta_{ijt} + \mu_t < 0 \qquad (8)$$

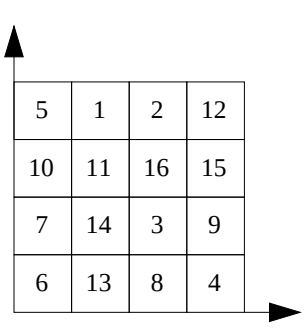
We use enumeration to create all feasible sequences for each Ω^t based on the initial bay layout. The initial configuration at period $t = 1$ is given. For periods $t = 2,\ldots,T-1$, possible configurations depend on the initial bay layout and on the relocations and retrievals of previous periods. We iterate from period 1 to period $T-1$ to create all feasible sequences. For each period t, all feasible sequences Ω^t are generated and attainable layouts for the next period are determined. If at least one sequence of Ω^t satisfies Constraint (8), the iteration stops after period t all sequences of Ω^t satisfying Constraint (8) are added to Ω^t_r.

In order to be feasible, a sequence has to meet several criteria. Obviously, containers can only be picked up from positions that they may occupy and can only be put into positions that may be free and are not suspended. In period t, only container $n = t$ may be retrieved. If the target container may be stored beneath other containers, those containers (and only those) have to be relocated within the bay to another stack. If two or more containers have to be relocated the LIFO order has to be respected. This means that if container n is stacked below container n' before the relocation, n cannot be stacked below container n' after the relocation.

We observed that the enumeration process is very slow since several millions of sequences have to be created. To speed up the subproblem, we present two bounding mechanisms to limit the number of possible relocations per sequences and hence the number of sequences to be created.

Bounding mechanism 1 This method uses lower and upper bounds on the total number of relocations to determine the maximum number of relocations per period. The initial bay layout at $t = 1$ makes it possible to determine a lower bound LB_t on the number or relocations needed to retrieve container $n = t$ for each period $t = 1,\ldots,T-1$. LB_t represents the number of containers that have to be relocated for sure to retrieve container t. We iterate from $t = 1$ to $T-1$. For each period t, LB_t is set to the number of containers placed above container t. Then, container t and all containers placed above it are deleted from the layout to avoid counting them twice. The consequence is that we have no information on relocated containers and their lower bounds have to be set to 0. In addition to LB_t, we can determine the minimum number of additional relocations AR_t that relocations at period t will cause in later periods. This may happen if a container has to be placed on a stack with at least one container with a higher priority. LB_t and AR_t are similar to the lower bounds used in Kim and Hong [7] and Zhang et al. [11].

Figure 3 illustrates the calculation of LB_t and AR_t. Containers 1 and 2 are at the topmost positions of their stacks and can be retrieved without any relocation ($LB_1 = 0$, $LB_2 = 0$).



Lower bound: (LB)		Add. relocations: (AR)	
t=1: 0	t=9: 0	**t=1: 0**	t=9: 0
t=2: 0	t=10: 0	t=2: 0	t=10: 0
t=3: 1	t=11: 0	t=3: 1	t=11: 0
t=4: 3	t=12: 0	t=4: 2	t=12: 0
t=5: 0	t=13: 1	t=5: 0	t=13: 0
t=6: 2	t=14: 0	t=6: 0	t=14: 0
t=7: 0	t=15: 0	t=7: 0	t=15: 0
t=8: 0		t=8: 0	

Figure 3: Calculation of lower bounds and additional relocations

To retrieve container 3, only container 16 has to be relocated since container 2 is already retrieved ($LB_3 = 1$). Each stack contains at least one container with a higher priority than container 16. Consequently, the relocation of container 16 will cause an additional relocation in a later period ($AR_3 = 1$). The lower bounds and the number of additional relocations for containers 4, 5, 6 and 13 are calculated accordingly. The lower bounds and the number of additional relocations for relocated containers 7, 8, 9, 10, 12, 14 and 15 are set to 0.

We also use the fact that the heuristic presented in Section 3.2 provides an upper bound UB on the total number of relocations over all periods $t = 1,\ldots,T-1$. LB_t, AR_t and UB enable us to determine the maximum number of admissible relocations per period k to obtain a solution with no more than UB relocations. Only sequences respecting Constraint (9) limiting the number of relocations at period k are generated.

$$c^s \leq UB - \sum_{t=1}^{k-1} LB_t - \sum_{t=k+1}^{T-1} LB_t - AR_k \quad \forall s \in \Omega^k, \text{k=1,...,T-1} \tag{9}$$

The limit of this bound is that LB_t and AR_t equal zero for all relocated containers. For initial bay layouts where containers with high priorities are stacked near the ground, most of the LB_t and AR_t will be zero. Thus, Constraint (9) may become $c^s \leq UB$ and has no or little impact.

Bounding mechanism 2 We simply impose that $c^s \leq$ initial stack height - 1 for all periods $t = 1,\ldots,T-1$. This bounding mechanism does not guarantee that the optimal solution for the restricted master problem is also optimal for the master problem. But, remember that only containers located above the container to be retrieved may be relocated. For that reason, few realistic initial bay layouts exist where more than initial stack height - 1 relocations are needed to solve the master problem optimally.

4 Preliminary results

We implemented an exact and a heuristic version of column generation. The exact column generation uses the bounding mechanism 1 to limit the number of generated variables. It guarantees that an optimal solution of the restricted master problem is also an optimal solution of the master problem. The heuristic column generation uses bounding mechanisms 1 and 2. In this case, the optimal solution of the restricted master problem is not necessarily an optimal solution of the master problem.

To evaluate the performance of both approaches we compare them with the performance of the binary model and the heuristic introduced by Caserta et al. [3]. We slightly modified their binary model by adding a constraint to make sure that the relocations are represented correctly and fixed several variables to zero to speed up the solution process. The heuristic is implemented according to their principles which were already presented in Section 3.2. The binary model and the restricted master problems are solved with Cplex 12.1. The heuristic and the exact and the heuristic subproblems are coded in C++. Experiments are carried out on the instances introduced by Caserta et al. [3]. Each instance H-W-x describes the initial configuration of the $N = W \cdot H$ containers in a bay with width W and initial stack height H. Five instances ($x = 1, \ldots, 5$) are given for each H-W. The maximum bay height is limited to $H + 2$. All experiments are run on a computer with Inter(R) Xeon(R) CPU clocked at 2.67GHz (dual core), 3.48GB RAM and operating with Windows XP Professional.

Results of this comparison are shown in Table 1. Column 1 identifies the instances. Column 2 shows the optimal solution (minimum number of relocations) of Caserta's binary model (MC) and column 3 the solution time. Columns 4 and 5 show the same results for Caserta's heuristic (HC), columns 6 and 7 for the relaxation of the binary model (rMC), columns 8 and 9 for the exact column generation (eCG) and columns 10 and 11 for the heuristic column generation (hCG). Instances that could not be solved with MC, eCG or hCG because Cplex run out of memory are marked by "mem". All instances for which the optimal binary solution is obtained are highlighted in bold for each solution method,. All instances for which the optimal solution value is obtained but variables are not binary are highlighted in italic for each solution method,.

The difficulty of an instance depends on the ratio of initial stack height to bay width and on the initial configuration of the bay. In general, instances with high initial stacks and instances where high priority containers are stacked close to the ground are more difficult to solve. For these instances, probabilities that containers have to be relocated one or several times are high. For MC, the number of variables depends on the size of the bay and on the number of containers. MC performs well for smaller instances 3-3 to 3-7 and 4-4, but less well for instances 3-8, 4-5 and 4-6 where execution times go up and memory problems arise. HC obtains almost always the optimal solution for instances with $H = 3$. Around 75% of the instances with $H = 4$ are solved optimally. The run time of the heuristic is always below 0.1 seconds. rMC provides very weak lower bounds on the optimal solution

Table 1: Comparison of the binary model, two column generation approaches and the heuristic

	MC		**HC**		**rMC**		**eCG**		**hCG**	
Inst.	Nb. rel	CPU time [s]	Nb. rel	CPU time [s]	Nb. rel	CPU time [s]	Nb. rel	CPU time [s]	Nb. rel	CPU time [s]
3-3-1	**6**	0.1	**6**	<0.1	*6.0*	0.1	**6.0**	0.2	**6.0**	0.1
3-3-2	**5**	0.2	**5**	<0.1	4.2	0.1	**5.0**	0.1	**5.0**	0.0
3-3-3	**2**	0.1	**2**	<0.1	*2.0*	0.1	**2.0**	0.0	**2.0**	0.0
3-3-4	**4**	0.1	**4**	<0.1	3.1	0.1	**4.0**	0.1	**4.0**	0.0
3-3-5	**1**	0.1	**1**	<0.1	**1.0**	0.1	**1.0**	0.0	**1.0**	0.0
3-4-1	**5**	0.3	**5**	<0.1	4.2	0.3	**5.0**	0.3	**5.0**	0.1
3-4-2	**3**	0.7	**3**	<0.1	**3.0**	0.3	**3.0**	0.0	**3.0**	0.0
3-4-3	**7**	0.6	**7**	<0.1	5.3	0.3	**7.0**	0.1	**7.0**	0.1
3-4-4	**5**	1.3	**5**	<0.1	4.3	0.4	**5.0**	0.2	**5.0**	0.1
3-4-5	**6**	1.2	**6**	<0.1	4.8	0.4	**6.0**	0.2	**6.0**	0.1
3-5-1	**6**	2.2	**6**	<0.1	5.2	1.0	**6.0**	0.4	**6.0**	0.1
3-5-2	**7**	3.7	**7**	<0.1	5.2	0.7	**7.0**	0.0	**7.0**	0.0
3-5-3	**8**	9.8	**8**	<0.1	5.9	1.1	**8.0**	0.9	**8.0**	0.2
3-5-4	**6**	7.3	**6**	<0.1	4.2	0.9	**6.0**	1.5	**6.0**	0.6
3-5-5	**9**	3.4	10	<0.1	*9.0*	0.8	*9.0*	382.5	*9.0*	5.1
3-6-1	**11**	44.3	**11**	<0.1	8.6	2.5	9.5	3848.6	9.5	69.6
3-6-2	**7**	14.4	**7**	<0.1	6.6	2.1	**7.0**	0.0	**7.0**	0.0
3-6-3	**11**	8.7	**11**	<0.1	8.7	1.9	mem	mem	**11.0**	11.9
3-6-4	**7**	18.3	**7**	<0.1	5.9	2.2	**7.0**	0.3	**7.0**	0.3
3-6-5	**4**	9.7	**4**	<0.1	3.1	1.8	**4.0**	0.0	**4.0**	0.0
3-7-1	**7**	8.9	**7**	<0.1	5.7	3.6	**7.0**	1.6	**7.0**	0.6
3-7-2	**10**	65.9	**10**	<0.1	6.8	5.8	**10.0**	5.2	**10.0**	0.6
3-7-3	**9**	10.4	**9**	<0.1	7.3	3.1	**9.0**	667.4	**9.0**	13.0
3-7-4	**8**	36.5	**8**	<0.1	6.3	5.4	**8.0**	1.0	**8.0**	1.0
3-7-5	**12**	36.8	**12**	<0.1	10.8	4.9	*12.0*	4836.1	**12.0**	86.8
3-8-1	**8**	34.9	**8**	<0.1	7.0	6.1	7.0	76.7	7.0	29.5
3-8-2	**10**	188.2	**10**	<0.1	5.3	14.5	**10.0**	15.9	**10.0**	0.8
3-8-3	mem	mem	**9**	<0.1	5.6	18.5	**9.0**	0.7	**9.0**	0.7
3-8-4	**10**	66.6	**10**	<0.1	7.9	6.9	*10.0*	29.8	**10.0**	3.0
3-8-5	mem	mem	**13**	<0.1	6.8	16.3	**13.0**	4878.1	**13.0**	620.4
4-4-1	**10**	13.2	**10**	<0.1	8.4	1.1	*10.0*	173.6	**10.0**	34.0
4-4-2	**10**	40.1	**10**	<0.1	5.7	1.8	**10.0**	4.8	**10.0**	4.3
4-4-3	**10**	6.6	11	<0.1	8.0	1.3	*10.0*	299.5	*10.0*	55.2
4-4-4	**7**	4.2	**7**	<0.1	6.4	1.2	**7.0**	0.0	**7.0**	0.0
4-4-5	**9**	7.6	10	<0.1	8.2	1.3	*9.0*	2.0	*9.0*	1.9
4-5-1	**16**	20279.8	**16**	<0.1	8.0	7.7	mem	mem	mem	mem
4-5-2	**10**	1561.6	11	<0.1	7.0	10.1	*10.0*	498.2	*10.0*	421.1
4-5-3	**13**	1144.7	**13**	<0.1	8.7	3.1	12.2	467.9	12.2	383.1
4-5-4	**8**	47.5	**8**	<0.1	6.8	2.8	**8.0**	16.5	**8.0**	1.5
4-5-5	**16**	19209.5	**16**	<0.1	10.5	5.5	mem	mem	mem	mem
4-6-1	mem	mem	18	<0.1	9.0	7.6	mem	mem	mem	mem
4-6-2	**8**	156.5	**8**	<0.1	5.0	13.6	7.0	2.8	7.0	2.8
4-6-3	**13**	49.5	**13**	<0.1	9.6	7.0	mem	mem	mem	mem
4-6-4	**14**	811.2	16	<0.1	10.8	11.2	mem	mem	mem	mem
4-6-5	mem	mem	**15**	<0.1	10.5	10.6	14.3	2548.7	14.3	413.4

of MC. The gap between the optimal solution of rMC and the optimal solution of MC may be as big as 50% and is almost never 0%. Like for MC, instances with large bays (3-8, 4-5 and 4-6) are the most difficult ones to solve.

The numbers of variables for hCG and eCG depend on the initial configuration and the initial stack height. The bounding mechanism 1 works fine for wide bays with low stacks, but not so well for narrow bays with high stacks. In general, eCG performs well for instance classes 3-3 to 3-8, but not so well for classes 4-4 to 4-6. But, for instances where high priority containers are stored at close to the ground, the bound provided by the bounding mechanism 1 is very weak and these instances need much time to be solved. Bounding mechanism 2 limits the influence of the initial bay layout and explains why these effects are less drastic for hCG. eCG finds the optimal binary solution for 60% of all instances and the gap equals zero for another 16% of the instances. For all tested instances, hCG obtains the same results than eCG and hence the optimal solution of the master problem. hCG finds the optimal binary solution for 69% of all instances and the gap equals zero for another 9% of the instances. We may thus conclude that hCG and eCG provide very strong lower bounds for the optimal solution of MC and that hCG outperforms eCG.

Remember that eCG and hCG perform well for wide bays with low stacks and MC for small bays with few containers. That is why eCG and hCG perform better for instances 3-3 to 3-8 and MC better for 4-4. For 4-5 and 4-6, hundreds of millions of variables have to be created for eCG and hCG and Cplex breaks down more often than for MC. But, for those instances that eCG and hCG solve, they perform better than MC.

Table 2 evaluates the performance of the two bounding mechanisms introduced in Section 3.3 and their impacts on column generation. The figures represent the average over all five instances of each class for columns 2 to 6 and the average over all instances solved with eCG and hCG for columns 7 to 10. Column 1 indicates the instance class. Column 2 shows the number of feasible sequences that are created if no bounding mechanism is applied. Columns 3 and 4 indicate how many sequences are created if the bounding mechanism 1 (BM1) is applied and the time needed to do so. Columns 5 and 6 show how many sequences are created if bounding mechanisms 1 and 2 (BM1+2) are applied and the execution time. Columns 7 and 8 show how many sequences are added to the restricted master problem and how often the subproblem is called for eCG. Columns 9 and 10 show the same parameters for hCG.

BM1 reduces the number of columns to be created drastically. It works especially well for instances 3-7 and 3-8 where the number of columns to be enumerated is reduced by 99%. Using BM2 in addition to BM1 reduces the number of created sequences even further. BM2 performs especially well for instances with $H = 4$. Using BM1+2, instead of BM1, speeds up the enumeration process considerably. Due to memory restrictions the variables cannot be registered and have to be recreated every time the subproblem is called. In addition, eCG needs more iterations that hCG to obtain the optimal solution for the restricted master problem. All these facts explain why hCG is solved faster than eCG, as we have seen in Table 1. Both, eCG and HCG add only a small percentage

Table 2: Comparison of bounding mechanisms BM1 and BM2

	No bounding	**BM1**		**BM 1 + 2**		**eCG**		**hCG**	
Inst. class	Generated columns	Generated columns	CPU time [s]	Generated columns	CPU time [s]	Added columns	Iterations	Added columns	Iterations
3-3	81.8	79.2	0.1	63.8	<0.1	23.6	4.8	22.0	4.6
3-4	1 594.6	424.0	<0.1	297.6	<0.1	78.8	7.2	76.0	7.2
3-5	1 342 497.8	355 589.8	3.9	19 148.2	0.2	9 569.2	14.2	1 577.8	10.6
3-6	30 694 056.4	16 597 984.4	263.1	93 570.0	0.8	198 631.3	18.0	9 519.3	17.3
3-7	127 412 801.2	1 003 561.6	21.8	233 469.8	2.0	112 695.8	31.2	1 762.4	19.0
3-8	225 395 905.5	1 185 319.0	23.0	235 450.4	2.6	204 112.0	30.0	58 846.8	23.2
4-4	312 643.4	285 191.0	7.7	69 759.8	1.6	34 847.2	16.8	9 789.6	17.4
4-5	283 559 725.8	232 843 810.4	3 604.8	11 615 308.6	127.6	26 558.3	32.7	31 597.0	36.0
4-6	628 145 323.8	416 072 882.0	8 519.5	26 613 126.4	364.3	42 063.0	62.5	15 091.0	53.5

For instances that we could not solve with eCG or hCG yet, more than 16 000 000 variables are created. We observe that eCG and hCG use only a small percentage of all generated sequences to solve the restricted master problem. To be able to solve bigger instances and to speed up the solution process further investigation to reduce the number of created variables has to be done.

5 Outlook

We presented an exact and a heuristic column generation approach for the container relocation problem. In both cases, the master problem represents the bay layout over time via configuration variables and movement variables. The configuration variables indicate the position of each container in the bay at each period. Each movement variable, called a sequence, represents a series of movements to retrieve one container and to relocate other containers. The subproblem enumerates all possible sequences. We use two very effective bounding mechanisms to reduce the number of created sequences and to speed up the solution process. First experiments are very promising since both approaches provide a very tight linear relaxation of the binary problem and need only a small number of sequences and iterations to obtain that solution. In around 70% of the cases, we obtain the optimal binary solution.

For bigger instances however, several millions of sequences have to be enumerated. This takes a long time and Cplex cannot handle all these variables. It is thus necessary to reduce the number of generated sequences to be able to solve bigger instances and to speed up the solution process. One possibility is to improve the quality of the lower bounds LB_t used in the bounding mechanism 1. In addition, column generation has to be embedded in a branching procedure to determine the optimal binary solution for the few cases where it does not provide the optimal binary solution.

References

[1] M. Caserta, S. Schwarze, and S. Voß. A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. In C. Cotta and P. Cowling, editors, *Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science*, volume 5482, pages 37–48. Springer, 2009.

[2] M. Caserta, S. Schwarze, and S. Voß. Container rehandling at maritime container terminals. In J. Böse, editor, *Handbook of Terminal Planning Operations*, volume 49, chapter 13, pages 247–269. Springer, 2011.

[3] M. Caserta, S. Schwarze, and S. Voß. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219:96–104, 2012.

[4] M. Caserta and S. Voß. Corridor selection and fine tuning for the corridor method. In T. Stützle, editor, *Learning and Intelligent Optimization, Lecture Notes in Computer Science*, volume 5851, pages 163–175. Springer, 2009.

[5] M. Caserta, S. Voß, and M. Sniedovich. Applying the corridor method to a blocks relocation problem. *OR Spectrum*, 33:915–929, 2011.

[6] F. Forster and A. Bortfeldt. A tree search procedure for the container relocation problem. *Computers & Operations Research*, 39:299–309, 2012.

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

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

[9] K.-C. Wu, R. Hernández, and C.-J. Ting. Applying tabu search for minimizing reshuffle operations at container yards. *Journal of the Eastern Asia Society for Transportation Studies*, 8, 2009.

[10] K.-C. Wu and C.-J. Ting. A beam search algorithm for minimizing reshuffle operations at container yards. In *International Conference on Logistics and Maritime Systems*, pages 703–710, September 15 - 17, Busan, Korea, 2010.

[11] H. Zhang, S. Guo, W. Zhu, A. Lim, and B. Cheang. An investigation of IDA* algorithms for the container relocation problem. In N. García-Pedrajas, F. Herrera, C. Fyfe, J. M. Benítez, and M. Ali, editors, *Trends in Applied Intelligent Systems, Lecture Notes in Computer Science*, volume 6096, pages 31–40. Springer, 2010.