A Hybrid Ant Colony and Branch-and-Cut Algorithm to Solve the Container Stacking Problem at Seaport Terminal

Ndèye Fatma Ndiaye
University of Le Havre
Laboratory of applied mathematics
25 Rue Philippe Lebon
76600 Le Havre cedex, France
Email: farlou@live.fr

Adnan Yassine
Superior institute of logistics studies
Quai Frissard, B.P. 1137
76063 Le Havre cedex, France
Email: adnan.yassine@univ-lehavre.fr

Ibrahima Diarrassouba
University Institute of Technology
Place Robert Schuman
76600 Le Havre cedex
France
Email: diarrasi@univ-lehavre.fr

***Abstract*—The container storage problem is one of the most studied issues regarding seaports. It is a relevant problem due to the fact that the effectiveness of a storage yard management affects the global productivity of the port. Therefore, various attempts were done in order to elaborate efficient decision support systems, which concern specific container terminals and specific transfer and handing equipments. Most of the existing proposed methods use heuristic or meta-heuristic algorithms because the NP-hardness of the container storage problem makes it difficult to solve using exact optimization methods mainly when there are a lot of containers. In this paper, we combine an exact resolution method (branch-and-cut) and a meta-heuristic algorithm (ant colony) in a hybrid ant colony and branch-and-cut algorithm (HACBC). Numerical simulations prove the efficiency and the effectiveness of our algorithm.**

***Keywords*–Ant colony algorithm; Branch-and-cut; Container storage problem; Hybridization; Mathematical modelling.**

I. INTRODUCTION

In a seaport, the container terminal manages all actions concerning containers. Generally, three types of containers are distinguished: outbound, inbound, and transshipment containers. All these containers are temporarily stacked in the container yard, before leaving the port. Outbound containers are brought by External Trucks (ETs), also picked up by the Straddle Carriers (SCs), which store them in their storage locations, and then loaded onto vessels. Inbound containers are unloaded from vessels by the Quai Cranes (QCs), transported to their storage locations by the SCs, and then recuperated later by ETs. Transshipment containers come to the port by ship and also leave the port by ship, after spending their dwell times in the storage yard.

Nowadays, the competition between ports is very high. Therefore, each of them tries to improve continuously the quality of its service in order to attract more customers. The most important criteria to measure service level, include the waiting time of ETs, which collect inbound containers. In fact, when an ET arrives at port and claims a specific container, it waits during all the time required to retrieve it. If the desired container is under others, it may be necessary to move firstly these containers. This kind of movements, named reshuffles, are unproductive and time consuming. Therefore, it is very important to optimally store containers. Another important criterion to measure the quality of service is the time required to unload ships. The importance of this factor is justified by the fact that it is more beneficial for both the port and the customers to shorten the stay of vessels. On one hand, it is better for the port authorities to quickly free the berths in order to allocate them to others incoming vessels. On other hand, generally shipowners rent vessels. Therefore, they tend to minimize the berthing durations in order to increase their profits. These two issues are addressed in this paper.

We consider a modern container terminal, which uses SCs instead of Internal Trucks (ITs). The advantage of a SC is the fact that it is able to lift and to store a container itself. Therefore, it is not necessary to use Yard Cranes (YCs). A storage yard is composed of several blocks. In order to enable the circulation of the SCs, each block is made up of several bays, which are separated by small spaces. In every bay, there are stacks wherein containers are stored. A stack must have a height inferior or equal to the limit fixed by the port authorities. Figure 1 shows an example of block wherein circulate straddle carriers.

Figure 1. Straddle carriers circulating in a containers yard

In this paper, we tackle the storage of inbound containers in a seaport terminal. We propose an efficient storage method, which enables to store the containers without causing no reshuffle. A linear mathematical model, which determines an

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

accurate storage location for each container is designed for this purpose. This mathematical model minimizes the total distance travelled by the straddle carriers from the quays to the storage yard. A branch-and-cut algorithm (BC-CSP) was proposed in [1] for the resolution of this problem. In this paper, we improve this algorithm by combining it with an ant colony algorithm.

The remainder of the paper is organized as follows: a literature review is given in Section II, a detailed description of the addressed problem is exposed in Section III, the mathematical model is explained in Section IV, the complexity of the problem is discussed in Section V, the branch-and-cut algorithm is itemized in Section VI, the ant colony algorithm is explained in Section VII, the hybrid ant colony and branch-and-cut algorithm is detailed in Section VIII, the numerical results are presented in Section IX, a conclusion is given in Section X.

II. Literature review

There are more papers addressing the storage of outbound containers than inbound containers. However, there are some papers that deal with both simultaneously. In [2], Zhang et al. considered in addition to these two categories of container, those that are in transition, that means the containers that are unloaded from some vessels and are waiting for being loaded onto other ships. They used the rolling-horizon approach to solve the storage space allocation problem. For each planing horizon, they solved the problem in two steps that are formulated as mathematical programs. In the first step, they determined the total number of containers that must be assigned to each block at a period so that the workload of loading and unloading of each vessel are balanced. Then, in the second step they determined the number of containers that must be associated to every vessel in order to minimize the total distance travelled to transport these containers from the quays to the storage blocks. In [3], Bazzazi et al. proposed a genetic algorithm to solve an extended version of the storage space allocation problem (SSAP). It consisted to allocate temporarily locations to the inbound and outbound containers in the storage yard according to their types (regular, empty, and refrigerated). They aimed to balance the workloads of the blocks with the goal to minimize the time required to store or to retrieve containers. In [4], Park et al. dealt with the planar storage location assignment problem (PSLAP), in which only planar movements were allowed. The purpose of the PSLAP was to store inbound and outbound containers so as to minimize the number of moving obstructive objects. The authors made a mathematical formulation of the PSLAP and proposed a genetic algorithm to solve it. In [5], Lee et al. combined the truck scheduling and the storage allocation problems. They considered inbound and outbound containers, and attempted to minimize the weighted sum of the total delay of requests and the total travel time of the yard trucks. For the numerical resolution, they proposed a hybrid insertion algorithm. In [6], Kozan et al. developed an iterative search algorithm by using a transfer model and an assignment model. At first, the algorithm determined cyclically the optimum storage locations for inbound and outbound containers, and secondly it found the corresponding handling schedule. They solved the problem by a genetic algorithm, a tabu search algorithm and a hybrid algorithm.

Concerning inbound containers, most of the papers dealt with the management of reshuffles. In [7], Sauri et al. proposed three different strategies to store inbound containers. The purpose of their work was to determine the best strategy that minimizes re-handles in an import container yard. For this, they developed a mathematical model based on probabilistic distribution functions to evaluate the number of reshuffles. In [8], Kim et al. considered a segregation strategy to store inbound containers. This method did not allow to place newly arriving containers over those that arrived earlier. Therefore, storage spaces are allocated to each vessel in order to minimize the number of expected reshuffles during the loading operations. In [9], Cao et al. proposed an integer programming model, which addressed the trucks scheduling and the storage of inbound containers. They minimized simultaneously the number of congestions, the waiting time of trucks, and the unloading time of containers. The authors designed a genetic algorithm to solve the model, and another heuristic algorithm, which outperformed their genetic algorithm. In [10], Yu et al. treated the storage problem of inbound containers in a modern automatic container terminal. They aimed to minimize the number of reshuffles in two steps. For this, they firstly resolved the block space allocation problem for the newly arriving inbound containers, and then, after the retrieving of some containers, they tackled the re-marshalling processes in order to re-organize the block space allocation. They suggested three mathematical models of storage containers, the first was a non-segregation model, the second was a single-period segregation model, and the third was a multiple-period segregation model. They conceived a convex cost network flow algorithm for the first and the second models, and a dynamic programming for the third. They found out that the re-marshalling problem is NP-hard, and then, they designed a heuristic algorithm to solve it. In [11], Moussi et al. considered a container terminal wherein reshuffles are not allowed. They proposed a new mathematical model to allocate storage spaces to inbound containers in such a way that no reshuffle will be necessary to retrieve them later. They designed a hybrid algorithm including genetic algorithm and simulated annealing to solve it. Ndiaye et al. strengthened that work by proposing in [1] a branch-and-cut algorithm, which is an exact optimization method, unlike the hybrid genetic and simulated annealing algorithm.

In most container terminals, the departure time of an inbound container is generally unknown. Kim et al. considered in [12] a container terminal, in which there is a limited free time storage for inbound containers, beyond which customers have to pay storage costs. The authors proposed a mathematical model to find the optimal price schedule.

Papers that dealt with the storage problem of outbound containers have generally different goals. In [13], Preston et al. proposed a container location model (CLM) to store outbound containers in a manner that minimised the time service of container ships. They designed a genetic algorithm for the numerical resolution. In [14], Kim et al. developed a dynamic programming model to determine storage locations for outbound containers according to their weights. They minimized the number of relocations expected during the loading operations of ships. They also made a decision tree using the set of optimal solutions to support real-time decisions. In [15], Chen et al. addressed in two steps the storage space allocation problem of outbound containers. In

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

the first step, they used a mixed integer programming model to calculate the number of yard bays and the number of locations in each of them. So, in the second step, they determined, for each container, the exact location where it will be stored. In [16], Woo et al. proposed a method to allocate storage spaces to groups of outbound containers. They reserved, for each group of containers that have the same attributes, a collection of adjacent stacks. At the end, the authors proposed a method to determine the necessary amount of storage spaces expected for all the outbound containers. In [17], Kim et al. gave two linear mathematical models to store outbound containers. In the first, they considered a direct transfer system. And then, in the second, they dealt with an indirect transfer system. They designed two heuristic algorithms to solve these models. The one was based on the duration-of-stay of containers, while they used the sub-gradient optimization technique in the other.

One of the few papers that dealt only with transshipment containers is that of Nishimura et al. [18]. The authors developed an optimization model to store temporarily transshipment containers in the storage yard, and proposed a heuristic based on Lagrangian relaxation method for the numerical resolution.

To the best of our knowledge, there is no paper that combines a meta-heuristic algorithm and an exact optimization method for the resolution of the container storage problem. Since the exact methods are not able to solve quickly large instances due to the NP-hardness of the problem, hybridization with meta-heuristic is a way to speed up the computation processes. So, in this paper, we exploit this idea by proposing a hybridization concerning a branch-and-cut algorithm and an ant colony algorithm.

III. Context

When a container ship arrives at port, the QCs unload the inbound containers and then place them on quays. After that, they are picked up by the SCs, which carry and store them in the container yard. The containers are picked up following the same order that they are unloaded from the ships. In order to avoid congestion at quays, which could increase the time required to unload the ships, we minimize the total distance travelled by the SCs between the quays and the container yard. In this study, we consider the following five hypotheses:
(1) reshuffles are not allowed,
(2) in each stack, the containers are stored following:
 (2.a) the same order that they are unloaded from the ships,
 (2.b) and the descending order of their departure times,
(3) in a stack, the containers have similar dimensions,
(4) we take into account the containers that are already present in the storage yard at the beginning of the current storage period,
(5) we do not exceed the maximum capacity of each stack.

Notice that the unloading order of the containers from a ship is decided by the port authorities before the arrival of this later at port. The role of such unloading plan is to ensure the stability of the ship during the unloading operations. However, the determination of the unloading plan and the storage plan (of inbound containers in the yard) are done separately, even if the results of the first problem are used for the resolution of the second. In this paper, we focus on the determination of the optimal storage plan of inbound containers in the yard.

IV. Mathematical modelling

In this section, we present a mathematical model that allocates an accurate storage location to every container. For this, we use the following indices, parameters, and decision variables.

Indices

k : container.
p : stack.
i : location in a stack.

Parameters

N_p : total number of stacks in the terminal.
c_p : number of available locations in the stack p.
r_p : size of the stack p, (20-feet, 40-feet, 45-feet, etc.).
t_p : departure time of the container that is at the top of the stack p at the beginning of the current storage period. It is equal to M if the stack is empty.
N : total number of inbound containers at quays.
T_k : departure time of the container k.
R_k : size of the container k, (20-feet, 40-feet, 45-feet, etc.).
d_p^k : distance between the stack p and the quay where is the inbound container k.
O_k : unloading order of the container k from ships.
M : a great integer.

Decision variables

$$x_{p,i}^k = \begin{cases} 1 & \text{If the location } i \text{ of the stack } p \text{ is allocated to the container } k. \\ 0 & \text{Otherwise.} \end{cases}$$

Model

$$\min \sum_{k=1}^{N} \sum_{p=1}^{N_p} \sum_{i=1}^{c_p} d_p^k x_{p,i}^k \qquad (1)$$

The objective function (1) minimises the total distance travelled by the straddle carriers between the quays and the storage yard.

$$\sum_{p=1}^{N_p} \sum_{i=1}^{c_p} x_{p,i}^k = 1, \; \forall\, k = 1, ..., N \qquad (2)$$

Constraint (2) ensures that each container is assigned to a single storage location.

$$\sum_{k=1}^{N} x_{p,i}^k \leq 1, \; \forall\, p = 1, ..., N_p, \; i = 1, ..., c_p \qquad (3)$$

Constraint (3) secures that several containers are not assigned simultaneously to a same storage location.

$$\sum_{p=1}^{N_p} \sum_{i=1}^{c_p} x_{p,i}^k = 0, \; \forall\, k = 1, ..., N \;:\; R_k \neq r_p \text{ or } T_k > t_p \qquad (4)$$

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Constraint (4) guarantees that a container can be assigned to a stack only if they are compatible. In other words, if the container and the stack have similar dimensions (20-feet, 40-feet, 45-feet, etc.), and if the departure time of the container is inferior or equal to that of the container that is already at the top of the stack (at the beginning of the new storage period).

$$\sum_{k=1}^{N}(N-O_k+1)x_{p,i}^{k} \geq \sum_{k=1}^{N}(N-O_k+1)x_{p,i+1}^{k} \quad \forall\, p=1,...,N_p,\ i=1,...,c_p-1 \tag{5}$$

Constraint (5) has two roles. The first is to enforce that, in every stack, the containers are stored following the ascending order of their unloading numbers. This avoids congestion at quays. The second is to secure that each stack is filled from bottom to top without omitting no location. This enables to satisfy the gravity's law, and excludes unrealisable assignments like the one that is shown in the example bellow.

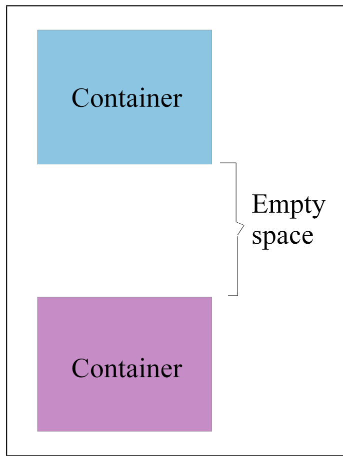


Figure 2. Unrealisable storage

$$\sum_{k=1}^{N} T_k x_{p,i}^{k} \geq \sum_{k=1}^{N} T_k x_{p,i+1}^{k} \quad \forall\, p=1,...,N_p;\ i=1,...,c_p-1 \tag{6}$$

Constraint (6) ensures that, in every stack, the containers are stored following the descending order of their departure times. This avoids reshuffles when extracting containers.

$$x_{p,i}^{k} \in \{0,1\}, \quad \forall\, p=1,...,N_p \quad \forall\, i=1,...,c_p,\ k=1,...,N \tag{7}$$

Constraint (7) states that the decision variables are boolean.

V. Complexity of the Problem

In this section, we study the complexity of the container storage problem (CSP). In particular, we show that it is equivalent to the bounded colouring problem (BCP). Therefore, it is NP-hard in the general case.

A. Some reminders about the BCP

Let us begin by recalling some concepts and definitions that will be useful for the following.

1) Preliminary notions: Let $G(V,E)$ be an undirected graph, V is the set of vertices and E is the set of edges.

G is a *comparability* graph if and only if there is a sequence of vertices $v_1,...,v_n$ of V such that for each (p,q,r) checking $1<p<q<r<n$, if $(v_p,v_q)\in E$ and $(v_q,v_r)\in E$, then $(v_p,v_r)\in E$.

A *co-comparability* graph is the complement of a comparability graph.

An undirected graph $G=(V,E)$ is a *permutation* graph if and only if there is a sequence of vertices $v_1,...,v_n$ of V and a permutation σ of the vertices such that: $\forall\, i,j\in\{1,...,n\}$, satisfying $1\leq i<j\leq n$, we have $(v_i,v_j)\in E$ if and only if $\sigma(i)>\sigma(j)$.

Theorem 1: A graph G is a permutation graph if and only if G and its complement are comparability graphs [19].

2) The bounded colouring problem: Given an undirected graph $G=(V,E)$, a set of s colours $l_1,...,l_s$, an integer H and a vector that gives the weight of assigning a colour l_i to a vertex of the graph. Solve the bounded colouring problem with minimum weight consists to determine a minimum weight colouring of G by using at most s colours in such a way that a colour is assigned to at most H vertices.

Theorem 2: The bounded colouring problem with minimum weight is NP-hard in the case of permutation graphs if $H\geq 6$ [20].

B. Case of storage where the stacks are empty at the beginning

In this section, we consider a case of storage where each stack of the storage yard is empty at the begin of the current storage period. We show that the CSP is NP-hard. For this, we introduce an undirected graph $G(N,O,T)=(V,E)$, which is constructed using an instance of the CSP, where N is the set of containers and O and T are two vectors that give respectively the unloading order and the departure time of each container. The graph G is constructed as follows. A vertex of the graph corresponds to a container. To simplify the notations, the index k is used to denote as well a container as the vertex that corresponds to it in the graph. There is an edge between two vertices k and k' if at least one of these two following conditions is satisfied:

- $O_k<O_{k'}$ and $T_k<T_{k'}$,
- $R_k\neq r_p$.

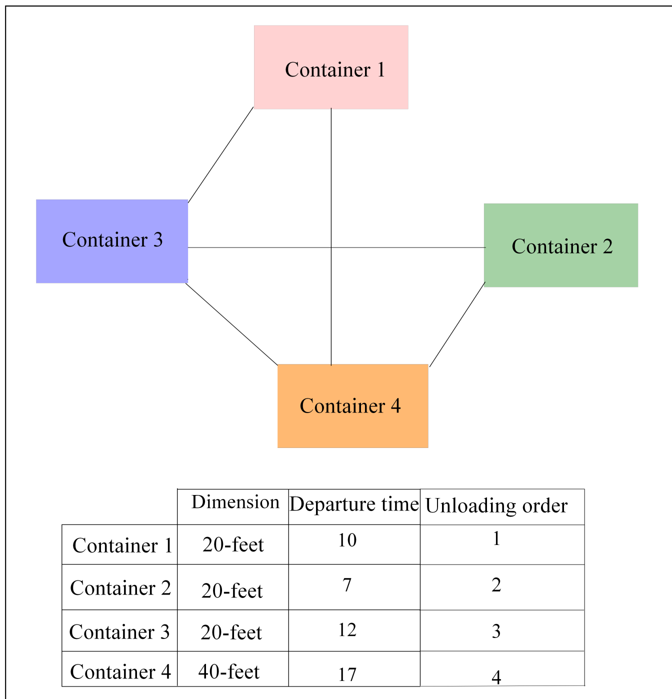
Figure 3. is an example of graph constructed using an instance of the CSP.

We have the following lemma.

Lemma 1: In the case where the containers have similar dimensions, the graph $G(N,O,T)$ obtained from an instance of the CSP is a permutation graph.

Proof: To prove the fact that the graph $G(N,O,T)$ is a permutation graph, it suffices to show that it is a comparability graph as well as its complement (see Theorem 1).

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org



	Dimension	Departure time	Unloading order
Container 1	20-feet	10	1
Container 2	20-feet	7	2
Container 3	20-feet	12	3
Container 4	40-feet	17	4

Figure 3. Graph constructed using an instance of the CSP.

Firstly, we show that $G(N, O, T)$ is a comparability graph. The vertices are ordered following the same order that the containers are unloaded from the ships. If two containers k and k' are unloaded from different ships at the same time (that is to say: $O_k = O_{k'}$), then the vertices k and k' are ordered in the ascending order of their departure times. If $O_k = O_{k'}$ and $T_k = T_{k'}$ then the vertices are ordered in the lexicographical order. Without loss of generality, we consider that the vertices are arranged in the order that is previously determined. Now, consider any three vertices k, k' and k'' of the graph, such that $k < k' < k''$, $(k, k') \in E$ and $(k', k'') \in E$. We will prove that necessarily $(k, k'') \in E$. As $(k, k') \in E$ and $(k', k'') \in E$, we have $O_k < O_{k'}$ and $T_k < T_{k'}$, and we have also $O_{k'} < O_{k''}$ and $T_{k'} < T_{k''}$. We thus obtain that $O_k < O_{k'} < O_{k''}$ and $T_k < T_{k'} < T_{k''}$, which implies that the graph $G(N, O, T)$ has an edge between the vertices k and k''. So $G(N, O, T)$ is a comparability graph.

At present, we will prove that the complement of $G(N, O, T)$, denoted $\bar{G}(N, O, T)$ is also a comparability graph. Firstly, notice that there is an edge between two vertices k and k' of $\bar{G}(N, O, T)$ if and only if there is no edge in $G(N, O, T)$ between k and k' (in other words $O_k < O_{k'}$ and $T_k > T_{k'}$). The vertices of $\bar{G}$ are ordered in the same order as those of G. As before, for any three vertices k, k', and k'' of the graph $\bar{G}(N, O, T)$, such that $k < k' < k''$, $(k, k') \in E$ and $(k', k'') \in E$, we have $O_k < O_{k'} < O_{k''}$ and $T_k > T_{k'} > T_{k''}$. So, $O_k < O_{k''}$ and $T_k > T_{k''}$, thus, there is an edge between k and k'' in $\bar{G}(N, O, T)$. Therefore, $\bar{G}(N, O, T)$ is also a comparability graph. ■

Now, it is easy to see that a solution of the container storage problem is a solution of the corresponding bounded colouring problem. In fact, a similar result is given in [21]. Consider an instance $ICSP = (N, O, T, N_p, H, r, R, d)$ of the CSP and the graph $G(N, O, T)$ associated to it, H is the maximum number of containers allowed in a stack. Consider a H-colouring of $G(N, O, T)$, which has s colours. Each colour of the bounded colouring problem is matched to a stack of the CSP. Indeed, as all vertices that have the same colour form a stable set, in other words they are not connected by any edges. Therefore, any two containers k and k' corresponding to two vertices of this stable set can be stored in a same stack because they satisfy these two inequalities: $O_k < O_{k'}$ and $T_k \geq T_{k'}$. The unloading order as well as the departure times of the containers that correspond to the vertices of a stable set are compatible, thereby they can be stored in a same stack if it has enough empty slots. In addition, there are at most H vertices in this stable set. So the number of containers assigned to the corresponding stack is inferior or equal to H. Therefore, a H-colouring corresponds to a valid assignment for the CSP. Similarly, it is easy to see that a solution of the CSP is a solution of the H-bounded colouring problem in the graph $G(N, O, T)$. We have the following lemma.

Lemma 2: Let $ICSP = (N, O, T, N_p, H, r, R, d)$ an instance of the container storage problem. The CSP has a solution for this instance if and only if the bounded H-colouring problem, considering the graph $G(N, O, T)$, has a solution.

Now, we give the main result of this section.

Theorem 3: The container storage problem is equivalent to the bounded colouring problem with minimum weight.

Proof: To establish this result, we prove that an instance of the CSP is equivalent to an instance of the BCP and vice versa. Let $ICSP = (N, O, T, N_p, H, r, R, d)$ an instance of the storage container problem and $G(N, O, T)$ the permutation graph associated to it. Consider $IBCP = (G(N, O, T), H, N_p, d)$ an instance of the BCP, which concerns the graph $G(N, O, T)$. N_p is the number of colours, H is the bound, and d is a matrix containing the weights. According to Lemma 2, a solution of the CSP is a solution of the BCP, and similarly a stack of the CSP corresponds to a colour of the BCP and vice versa. It follows then that the cost d_p^k of assigning a container $k \in N$ to the stack $p \in N_p$, is the same as the assignment of the vertex k to the colour corresponding to the stack p. So, the cost of a H-colouring of the graph $G(N, O, T)$ is similar to the cost of the solution of the corresponding CSP and vice versa. Therefore, we can find the optimal solution of the CSP if and only if we find the optimal solution of BCP. ■

According to Theorem 2, the bounded colouring problem is NP-hard in the case of permutation graphs if $H \geq 6$. Therefore, it follows from Theorem 3 that the CSP is NP-hard if $H \geq 6$.

Corollary 1: In the case where the containers have similar dimensions, the container storage problem is NP-hard if the maximum capacity of each stack is superior or equal to six.

If the containers have different sizes, we suppose that they are divided into groups, each having similar containers. Similarly, the stack are also divided into groups, each containing stacks that have equal measures. So, the CSP can be solved by considering separately several sub-problems, each consisting of allocating storage spaces to a group of containers that have similar sizes. For example, if there are three sizes of containers (20-feet, 40-feet, and 45-feet), we have three groups of container (K_{20}, K_{40}, K_{45}) and three groups of stack (P_{20}, P_{40}, P_{45}). In this example, to solve

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

the CPS, we can solve three sub-problems (CSP$_{20}$, CSP$_{40}$, CSP$_{45}$) by considering respectively (K_{20}, P_{20}), (K_{40}, P_{40}), and (K_{45}, P_{45}). So, the solutions of these sub-problems constitute the solution of the initial problem, as shown in Figure 4.

Since there is one size of container in each sub-problem, the corresponding graphs are permutation graphs (see Lemma 1). So, the container storage problem stills NP-hard even if the sizes of the containers are not similar.

Corollary 2: In the case where there are several dimensions of container, the container storage problem is NP-hard if the maximum number of containers allowed in a stack is superior or equal to six.

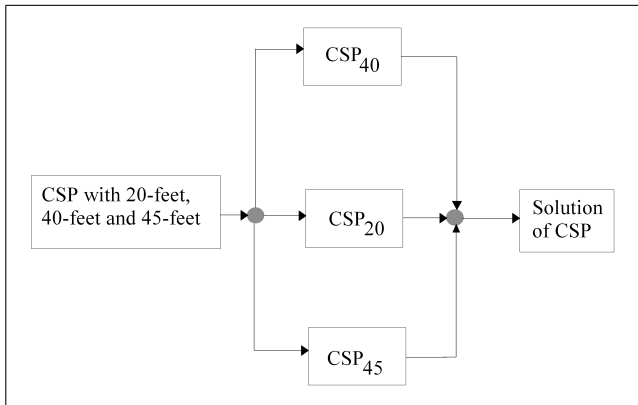


Figure 4. Case with three measures

C. Case of storage where some stacks are not empty at the beginning

In the case where some stacks are not empty at the beginning of the current storage period, the CSP can be formulated as a colouring problem with capacity (CPC), which is a generalization of the bounded colouring problem [21]. In the CPC, each colour p has a capacity c_p, which is the maximum number of nodes that can receive this colour. Different colours may have different capacities.

If we have an instance of the CSP, in which each stack has a capacity c_p, we can construct the corresponding graph and solve the CPC by considering that each colour can be used at most c_p times.

VI. Branch-and-cut algorithm

In the branch-and-cut algorithm, we consider the graph depicted in Section V-B, in which each node represents a container, and each edge connects two containers (nodes) that does not have similar dimensions or two containers that have conflicting arrival orders and departure times. So, as we know the fact that two adjacent containers cannot be stored into a same stack, we exploit this propriety in the branch-and-cut algorithm and we use the simplified mathematical model that follows.

The former decision variable $x_{p,i}^k$ is simplified and becomes x_p^k, which is defined as follows.

$$x_p^k = \begin{cases} 1 & \text{If the container k is assigned to the stack } p \\ 0 & \text{Otherwise} \end{cases}$$

As can be seen, this decision variable specifies the stack that is allocated to each container, but it does not point out the exact storage location of a container in a stack. This does not cause any problems, because the method that is used to construct the graph ensures that the unloading order and the departure times of the containers that are assigned to a same stack are compatible.

The new mathematical model is:

$$Minimize \sum_{k=1}^{N} \sum_{p=1}^{N_p} d_p^k x_p^k \qquad (8)$$

The objective function (8) minimizes the total distance travelled by the straddle carriers between the ships and the container yard.

$$\sum_{p=1}^{N_p} x_p^k = 1, \quad \forall\, k = 1, ..., N \qquad (9)$$

Constraint (9) requires that each container is assigned to a single stack.

$$x_p^k + x_p^{k'} \leq 1, \quad \forall\, (k, k') \in E,\ p = 1, ..., N_p \qquad (10)$$

Constraint (10) ensures that the containers of each stack can be arranged following the same order that they were unloaded from ships, and the decreasing order of their departure times.

$$\sum_{k=1}^{N} x_p^k \leq c_p, \quad \forall\, p = 1, ..., N_p \qquad (11)$$

Constraint (11) enforces that the capacity of each stack is not exceeded.

$$\sum_{k=1}^{N} x_p^k = 0, \quad \forall\, p = 1, ..., N_p : T_k > t_p \text{ or } R_k \neq r_p \qquad (12)$$

Constraint (12) secures that each container can be assigned only to a compatible stack.

$$x_p^k \in \{0, 1\}, \quad \forall\, k = 1, ..., N,\ p = 1, ..., N_p \qquad (13)$$

Constraint (13) states that the decision variables are boolean.

A. Description of the algorithm

The branch-and-cut algorithm is an exact resolution method, which combines the branch-and-bound method and the cutting plane method. Each of them proceeds by solving a sequence of relaxations of the mixed integer linear problem. The cutting plane methods improve the relaxation of a problem in order to ameliorate the approximation, while the branch-and-bound methods use a well known approach named "*divide and conquer*".

The branch-and-cut algorithm uses a search tree whose root node is the integer problem that needs to be solved. The

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

other nodes of the search tree are created sequentially by partitioning the search space, in other words, by creating new branches. The major difference between the branch-and-cut and the branch-and-bound is the fact that the former uses valid inequalities (cutting planes) to improve the solution found at each node of the search tree before performing branching. Notice that, a *valid inequality* is an inequality that must be satisfied by each solution of the mixed integer problem, but that is not necessarily satisfied by the solutions of the relaxations.

To perform a branch-and-cut, it is necessary to have these elements: one or several valid inequalities, a relaxation method, a technique to find an upper bound, a branching rule, and a method to look for violated valid inequalities (this method is called separation method).

In the following, we will explain them, before describing the algorithm.

1) Creation of a valid inequality: Let k a node (container) of the graph, such that $1 \leq k \leq N$. $N(k)$ the set of the neighbours of k, and p a colour (stack) such that $1 \leq p \leq N_p$.

After calculating the sum of constraints (10) in the neighbourhood $N(k)$, we get the following valid inequality named *neighbourhood inequality*.

$$\sum_{k' \in N(k)} x_p^{k'} + |N(k)| x_p^k \leq |N(k)| \tag{14}$$

Proposition 1: For a solution of the integer program, the inequalities (10) and (14) are equivalent.

Proof: It suffices to prove that (14) implies (10), because the reverse is highlighted by the definition of (14).
As x_p^k is a binary variable, it can be equal to either 0 or 1.
• If $x_p^k = 1$, then $\sum_{k' \in N(k)} x_p^{k'} = 0$. Therefore, for all k' neighbour of k, we have $x_p^{k'} = 0$. Thereby, $x_p^k + x_p^{k'} = 1, \forall k' \in N(k)$.
• If $x_p^k = 0$, then $\sum_{k' \in N(k)} x_p^{k'} \leq |N(k)|$, which means that for all k' belonging to $N(k)$, $x_p^{k'}$ can be equal to either 0 or 1. Thus, $x_p^{k'} + x_p^k \leq 1, \forall k' \in N(k)$. ∎

2) Relaxation of the problem: Generally, in a branch-and-cut algorithm, the integrity constraints (6) are relaxed by replacing them with ($x_p^k \geq 0, \forall k = 1, ..., N; \; p = 1, ..., N_p$). However, in our case, we go further by removing to the relaxed problem the constraint (10). This does not affect the optimality of the final solution, because at each node of the search tree, valid inequalities (14) are added to the program, in order to ensure the correctness of the solutions.

3) Preprocessing: The number of variables increases depending on the number of stacks (N_p) and the number of containers (N). In the case where all stacks were empty at the beginning of the storage period, if all containers are discharged from a same vessel, we can reduce the number of variables, because the containers are equidistant to the stacks. In such case, we only use the N stacks that are closer to the quay. This allows to significantly reduce the number of variables and to speed up the computation.

4) Upper bound: To find an upper bound, we solve the bounded vertex colouring problem applied on the graph defined in Section V-B. Each colour corresponds to a stack. We use a heuristic algorithm, which colours vertices one by one following the descending order of their number of uncoloured neighbours. For each vertex, it chooses among the admissible colours the one that fits to the nearest stack. The eligible colours are those that are not assigned to a vertex that is a neighbour of the considering vertex, and correspond to the stacks that are not full and satisfy the compatibility constraints (12). Whenever a vertex is coloured, the number of empty slot of the stack corresponding to the used colour is reduced.

5) Branching rule: We use the classical branching rule. At each node of the search tree, we create two branches by rounding the most fractional variable. Let x_p^k this variable. We put $x_p^k = 0$ in a branch, it means that the container k will not be assigned to the stack p in this branch. Then, in the other branch, we put $x_p^k = 1$, which means that the container k will be inevitably assigned to the stack p in this branch.

The most fractional variable is the one that is mostly half-way to the largest integer that is not greater than it and the smallest integer that is not lesser than it. For example, if we have $\{0.25, 0.45, 0.75\}$, the most fractional variable is 0.45.

To search the most fractional variable, we use this following algorithm.

1. $select = 1$;
2. For ($k = 1, ..., N$) do
    2.a. For ($p = 1, ..., N_p$) do
        2.a.1. If ($|\frac{1}{2} + \lfloor x_p^k \rfloor - x_p^k| < select$) then $select = x_p^k$
        2.a.2 End If
    2.b. End For
3. End For

6) Separation method: At each node of the search tree, before creating branches, we use a simple algorithm to look for neighbourhood inequalities that are violated. To do this, we treat one by one each variable that is superior to 0.5 in the optimal solution of the current node. Let x_p^k one of these variables and S an integer that is initialized to zero. We calculate the number $|N(k)|$ of neighbours of the vertex k. Then, we add to S the value of x_p^k multiplied by $|N(k)|$. After that, we seek every variable $x_{p'}^{k'}$, which is such that k and k' are neighbours and $p = p'$, and we add the sum of their values to S. If $S > |N(k)|$, there is a violated inequality; therefore, we add to the sub-problem a constraint to avoid this.

7) Algorithm: The branch-and-cut algorithm that we propose to solve the container storage problem (BC-CSP) is as follows.

a. *Initialization:* We note P^0 the initial integer program. And then, we initialize the search tree $T = \{P^0\}$. After that, we use the algorithm depicted in Section VI-A3 to find an upper bound **UB**. If no solution is found, we set $\mathbf{UB} = +\infty$.

b. *Stop criterion:* If $T = \emptyset$, the optimal solution is the one whose value equals **UB**. If $\mathbf{UB} = +\infty$, there is no realizable

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

solution.

c. *Selection:* Choose a node P^l of T. This node will be removed from T after being explored.

d. *Relaxation:* Relax P^l, and then, solve it using the CPLEX solver. If there is no solution, set $\mathbf{LB}_l = +\infty$, after that, go to Step f. Otherwise, denominate $\mathbf{S}^{Rl}$ the optimal solution of the relaxation of P^l, and then, use its value to update $\mathbf{LB}_l$.

e. *Separation:* Seek all the valid inequalities that are violated by $\mathbf{S}^{Rl}$, add them to the relaxation of P^l, and then, go back to Step d.

f. *Fathoming and Pruning:* If $\mathbf{LB}_l \geq \mathbf{UB}$, then go back to Step b. If $\mathbf{LB}_l < \mathbf{UB}$ and $\mathbf{S}^{Rl}$ is a realizable integer solution, then update **UB = LB**$_l$ and remove from T every node j that satisfies $\mathbf{LB}_j \geq \mathbf{UB}$, after that, go back to Step b.

g. *Branching:* If $\mathbf{S}^{Rl}$ is fractional, then use the most fractional variable to perform a branching, in order to get two new sub-problems $P^{l,1}$ and $P^{l,2}$, after that, add them to T.

B. Example

Suppose that we need to store five containers in three empty stacks. The maximum number of containers allowed in a stack is equal to 3. We want to find the optimal storage plan, which does not lead to reshuffles and, which minimizes the total distance travelled by the straddle carriers between the quays and the storage yard. Table I and Table II show the distances and the characteristics of the containers, respectively. We consider that each container and each stack measures 20-feet.

TABLE I. Distances.

p	d_p^1	d_p^2	d_p^3	d_p^4	d_p^5
1	105	378	205	378	400
2	118	391	118	291	413
3	106	378	165	332	314

d_p^k is the distance between the stack p and the quay where is the container k.

TABLE II. Characteristics of the containers.

Container	*Departure time*	*Unloading order*
1	10	1
2	13	2
3	8	3
4	16	4
5	10	5

From these data we have the graph represented in Figure 5.

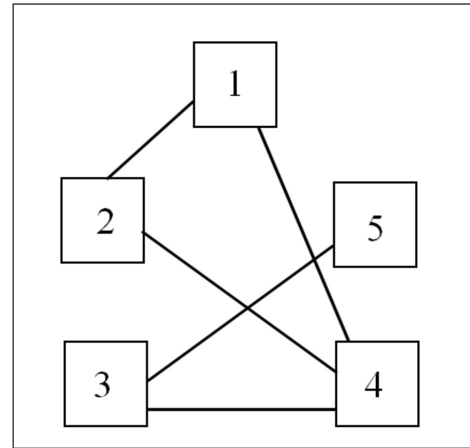


Figure 5. Graph constructed using the instance depicted in the Table II.

Firstly, we use the algorithm described in Section VI-A4 to look for an upper bound (in other words an integer solution that is not necessarily optimal). For this, we begin by the container 4, because it has most neighbours, we assign it to the stack 2, which is the nearest. After that, we remark that each of the remaining containers has one unassigned neighbour. Thereby, we realise these assignments, following the same order: container 1 to the stack 1, container 3 to stack 3, container 2 to stack 3, and container 5 to stack 1. The obtained upper bound is **UB**=1339, which equals $d_2^4 + d_1^1 + d_3^3 + d_3^2 + d_1^5$.

From the graph represented in Figure 5 and the data of Table I and Table II, we have the following linear integer program.

$$P^0 = \begin{cases} \text{Minimize} \\ d: 105x_1^1 + 118x_2^1 + 106x_3^1 + 378x_1^2 + 391x_2^2 + \\ \quad 378x_3^2 + 205x_1^3 + 118x_2^3 + 165x_3^3 + 378x_1^4 + \\ \quad 291x_2^4 + 332x_3^4 + 400x_1^5 + 413x_2^5 + 314x_3^5 \\ \text{subject to} \\ (1): x_1^1 + x_2^1 + x_3^1 = 1 \\ (2): x_1^2 + x_2^2 + x_3^2 = 1 \\ (3): x_1^3 + x_2^3 + x_3^3 = 1 \\ (4): x_1^4 + x_2^4 + x_3^4 = 1 \\ (5): x_1^5 + x_2^5 + x_3^5 = 1 \\ (6): x_1^1 + x_1^4 \leq 1 \\ (7): x_2^1 + x_2^4 \leq 1 \\ (8): x_3^1 + x_3^4 \leq 1 \\ (9): x_1^1 + x_1^2 \leq 1 \\ (10): x_2^1 + x_2^2 \leq 1 \\ (11): x_3^1 + x_3^2 \leq 1 \\ (12): x_1^2 + x_1^4 \leq 1 \\ (13): x_2^2 + x_2^4 \leq 1 \\ (14): x_3^2 + x_3^4 \leq 1 \\ (15): x_1^3 + x_1^4 \leq 1 \\ (16): x_2^3 + x_2^4 \leq 1 \\ (17): x_3^3 + x_3^4 \leq 1 \\ (18): x_1^3 + x_1^5 \leq 1 \\ (19): x_2^3 + x_2^5 \leq 1 \\ (20): x_3^3 + x_3^5 \leq 1 \\ (21): x_1^1 + x_1^2 + x_1^3 + x_1^4 + x_1^5 \leq 3 \\ (22): x_2^1 + x_2^2 + x_2^3 + x_2^4 + x_2^5 \leq 3 \end{cases}$$

$$\begin{cases} (23): x_3^1 + x_3^2 + x_3^3 + x_3^4 + x_3^5 \leq 3 \\ (24): x_p^k \in \{0,1\},\ \forall\, k = 1,...,5;\ p = 1,2,3 \end{cases}$$

We initialize the search tree $T = \{P^0\}$. After that, we relax P^0, so we get R^0 that is as follows.

$$R^0 = \begin{cases} \text{Minimize} \\ d: 105x_1^1 + 118x_2^1 + 106x_3^1 + 378x_1^2 + 391x_2^2 + \\ \quad 378x_3^2 + 205x_1^3 + 118x_2^3 + 165x_3^3 + 378x_1^4 + \\ \quad 291x_2^4 + 332x_3^4 + 400x_1^5 + 413x_2^5 + 314x_3^5 \\ \text{subject to} \\ (1): x_1^1 + x_2^1 + x_3^1 = 1 \\ (2): x_1^2 + x_2^2 + x_3^2 = 1 \\ (3): x_1^3 + x_2^3 + x_3^3 = 1 \\ (4): x_1^4 + x_2^4 + x_3^4 = 1 \\ (5): x_1^5 + x_2^5 + x_3^5 = 1 \\ (21): x_1^1 + x_1^2 + x_1^3 + x_1^4 + x_1^5 \leq 3 \\ (22): x_2^1 + x_2^2 + x_2^3 + x_2^4 + x_2^5 \leq 3 \\ (23): x_3^1 + x_3^2 + x_3^3 + x_3^4 + x_3^5 \leq 3 \\ (24): x_p^k \geq 0,\ \forall\, k = 1,...,5;\ p = 1,2,3 \end{cases}$$

After that, we solve R^0, using the CPLEX solver (version 12.5). And then, we get ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 1, x_2^2 = 0, x_1^3 = 0, x_2^3 = 1, x_1^4 = 0, x_2^4 = 1, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0, x_3^2 = 0, x_3^3 = 0, x_3^4 = 0, x_3^5 = 1$), which equals 1206. This is not a realizable solution, because it does not satisfy the following valid inequalities.

$(25): x_1^2 + x_1^4 + 2x_1^1 <= 2$
$(26): x_1^1 + x_1^4 + 2x_1^2 <= 2$
$(27): x_2^4 + x_2^5 + 2x_2^3 <= 2$
$(28): x_2^1 + x_2^2 + x_2^3 + 3x_2^4 <= 3$

Then, we add the inequalities (25), (26), (27), and (28) to R^0. After that, we solve it again, and then, we obtain ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 0, x_2^2 = 0, x_1^3 = 0, x_2^3 = 0.6, x_1^4 = 0, x_2^4 = 0.8, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0, x_3^2 = 1, x_3^3 = 0.4, x_3^4 = 0.2, x_3^5 = 0$), which equals 1233. This solution is fractional. In addition, it does not satisfy the following valid inequalities.

$(29): x_3^1 + x_3^4 + 2x_3^2 <= 2$
$(30): x_3^3 + x_3^5 <= 1$

We add the inequalities (29) and (30) to R^0. And then, we solve it again. So, we get ($x_1^1 = 0.666666666666667, x_2^1 = 0, x_1^2 = 0.666666666666667, x_2^2 = 0, x_1^3 = 0, x_2^3 = 1, x_1^4 = 0, x_2^4 = 0, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0.333333333333333, x_3^2 = 0.333333333333333, x_3^3 = 0, x_3^4 = 1, x_3^5 = 1$), which equals 1247.33333333333. This solution violates the following valid inequality.

$(31): x_3^1 + x_3^2 + x_3^3 + 3x_3^4 <= 3$

After adding the inequality (31) to R^0, we solve it again. And then, we get ($x_1^1 = 0.75, x_2^1 = 0, x_1^2 = 0.5, x_2^2 = 0, x_1^3 = 0.125, x_2^3 = 0.875, x_1^4 = 0, x_2^4 = 0.25, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0.25, x_3^2 = 0.5, x_3^3 = 0, x_3^4 = 0.75, x_3^5 = 1$), which equals 1247.875. This solution does not violate any valid inequalities, but it is not integer. So, we use the variable x_1^2 to do a branching. And then, we get two sub-problems $P^{00} = P^0 \cup \{x_1^2 = 0\}$ and $P^{01} = P^0 \cup \{x_1^2 = 1\}$. After that, we add the new sub-problems to T, and we remove P^0. So, we have $T = \{P^{00}, P^{01}\}$.

Since T is not yet empty, we continuous the algorithm. So, we select P^{00} from T. After that, we relax it, and we obtain R^{00}. Then, we solve R^{00} using CPLEX, and we obtain ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 0, x_2^2 = 0.125, x_1^3 = 0.375, x_2^3 = 0.625, x_1^4 = 0, x_2^4 = 0.75, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0, x_3^2 = 0.875, x_3^3 = 0, x_3^4 = 0.25, x_3^5 = 1$), which equals 1250.5. This solution does not violate any valid inequalities. So, we use x_1^3 to do a branching. Then, we get two new sub-problems $P^{000} = P^{00} \cup \{x_1^3 = 0\}$ and $P^{001} = P^{00} \cup \{x_1^3 = 1\}$. We add these sub-problems to T, and we remove P^{00}. So, we have $T = \{P^{01}, P^{000}, P^{001}\}$.

We select P^{01}. After that, we relax and solve it. And then, we obtain ($x_1^1 = 0, x_2^1 = 0, x_1^2 = 1, x_2^2 = 0, x_1^3 = 0.166666666666667, x_2^3 = 0.833333333333333, x_1^4 = 0, x_2^4 = 0.333333333333333, x_1^5 = 0, x_2^5 = 0, x_3^1 = 1, x_3^2 = 0, x_3^3 = 0, x_3^4 = 0.666666666666667, x_3^5 = 1$), which equals 1248.83333333333. This solution violates the following valid inequality.

$(32): x_3^2 + x_3^4 + 2x_3^1 <= 1$

We add the inequality (32) to R^{01}. And then, we solve it again. After that, we get ($x_1^1 = 0, x_2^1 = 0.125, x_1^2 = 1, x_2^2 = 0, x_1^3 = 0.375, x_2^3 = 0.625, x_1^4 = 0, x_2^4 = 0.75, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0.875, x_3^2 = 0, x_3^3 = 0, x_3^4 = 0.25, x_3^5 = 1$), which equals 1251.375. This solution does not violate any valid inequalities. So, we use x_1^3 to do a branching. Two sub-problems are then created and added to T, $P^{010} = P^{01} \cup \{x_1^3 = 0\}$ and $P^{011} = P^{01} \cup \{x_1^3 = 1\}$. We remove P^{01} from T, which becomes $T = \{P^{000}, P^{001}, P^{010}, P^{011}\}$.

We select P^{000}. And then, we relax and solve it. After that, we get ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 0, x_2^2 = 0.375, x_1^3 = 0, x_2^3 = 0.875, x_1^4 = 0, x_2^4 = 0.25, x_1^5 = 0.125, x_2^5 = 0, x_3^1 = 0, x_3^2 = 0.625, x_3^3 = 0.125, x_3^4 = 0.75, x_3^5 = 0.875$), which equals 1258.25. This solution does not violate any valid inequalities. So, we use x_2^2 to perform a branching. Then, we get two new sub-problems $P^{0000} = P^{000} \cup \{x_2^2 = 0\}$ and $P^{0001} = P^{000} \cup \{x_2^2 = 1\}$. We remove P^{000} from T. After that, we add to T the new sub-problems. So, we have $T = \{P^{001}, P^{010}, P^{011}, P^{0000}, P^{0001}\}$.

We select P^{001}. After that, we relax and solve it. Then, we get ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 0, x_2^2 = 0, x_1^3 = 1, x_2^3 = 0, x_1^4 = 0, x_2^4 = 1, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0, x_3^2 = 1, x_3^3 = 0, x_3^4 = 0, x_3^5 = 1$), which equals 1293. This solution is integer and does not violate any valid inequalities. So, we update the upper bound **UB**=1293. And then, we remove P^{001} from T, which becomes $T = \{P^{010}, P^{011}, P^{0000}, P^{0001}\}$.

We select P^{010}. Then, we relax and solve it. After that, we get ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 0, x_2^2 = 0, x_1^3 = 1, x_2^3 = 0, x_1^4 = 0, x_2^4 = 1, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0, x_3^2 = 1, x_3^3 = 0, x_3^4 = 0, x_3^5 = 1$), which equals 1293. This is an integer solution, which does not violate any valid inequalities. We remove

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

P^{010} from T, which becomes $T = \{P^{011}, P^{0000}, P^{0001}\}$.

We select P^{011}. After that, we relax and solve it. Then, we get ($x_1^1 = 0, x_2^1 = 0, x_1^2 = 1, x_2^2 = 0, x_1^3 = 1, x_2^3 = 0, x_1^4 = 0, x_2^4 = 1, x_1^5 = 0, x_2^5 = 0, x_3^1 = 1, x_3^2 = 0, x_3^3 = 0, x_3^4 = 0, x_3^5 = 1$), which equals 1294. We remove P^{011} from T, because it gives a solution that has a value superior to the upper bound. So, we have $T = \{P^{0000}, P^{0001}\}$.

We select P^{0000}. After that, we relax and solve it. Then, we get ($x_1^1 = 0.875, x_2^1 = 0.125, x_1^2 = 0, x_2^2 = 0, x_1^3 = 0, x_2^3 = 0.625, x_1^4 = 0.25, x_2^4 = 0.75, x_1^5 = 0.375, x_2^5 = 0, x_3^1 = 0, x_3^2 = 1, x_3^3 = 0.375, x_3^4 = 0, x_3^5 = 0.625$), which equals 1279.25. This solution does not violate any valid inequalities. So, we use x_2^3 to do a branching. Then, we get two new sub-problems: $P^{00000} = P^{0000} \cup \{x_2^3 = 0\}$ and $P^{00001} = P^{0000} \cup \{x_2^3 = 1\}$. After that, we update the list of active nodes, $T = \{P^{0001}, P^{00000}, P^{00001}\}$.

We select P^{0001}. Then, we relax and solve it. After that, we get ($x_1^1 = 1, x_2^1 = 0, x_1^2 = 0, x_2^2 = 1, x_1^3 = 0, x_2^3 = 1, x_1^4 = 0, x_2^4 = 0, x_1^5 = 0, x_2^5 = 0, x_3^1 = 0, x_3^2 = 0, x_3^3 = 0, x_3^4 = 1, x_3^5 = 1$), which equals 1260. This solution is integer and does not violate any valid inequalities. So, we update the upper bound, **UB**=1260. And then, we remove P^{0001}, P^{00000}, and P^{00001} from T. Notice that P^{00000} and P^{00001} are not explored because their lower bound (1279.25) is superior to the general upper bound (1260). This is the end of the algorithm, because $T = \emptyset$.

The search tree corresponding to that example is represented in Figure 6. The red nodes are those that are pruned, while the green nodes are the ones that allow to update the upper bound (in other words those that give integer solution, which is better than the current one). The order, in which the nodes are explored is specified by the blue writings.

VII. ANT COLONY ALGORITHM

The first ant colony algorithm is invented by Dorigo et al. [22]. They were inspired by the behaviour of the natural ants when they are looking for food. These animals communicate indirectly via a natural substance named pheromone, in order to discover the shortest path between their anthill and a location where there is food. This substance is continuously deposited on the travelled ways. Therefore, since the short paths lead more quickly to the food, the pheromone will be accumulated there more quickly. So, they will be more preferable. In addition to this, the pheromone tends to disappear on the longer paths due to the evaporation.

To apply an ant colony algorithm to a problem, it is necessary to define how to represent a solution. So, in the following, we firstly specify how we encode our solutions, before detailing the ant colony algorithm that we propose to solve the container storage problem.

A. Method to represent a solution

In the ant colony algorithm, we represent a solution as an array that has two rows. The containers are written in the first row, while the stacks are noted in the second. The number of columns in the solution is equal to the number of containers that need to be stored. The following example represents a solution, in which six containers are assigned to three stacks.

Containers →	4	2	3	1	6	5
Stacks →	1	3	2	2	3	3

Figure 7. Example of solution

This solution corresponds to the following assignment

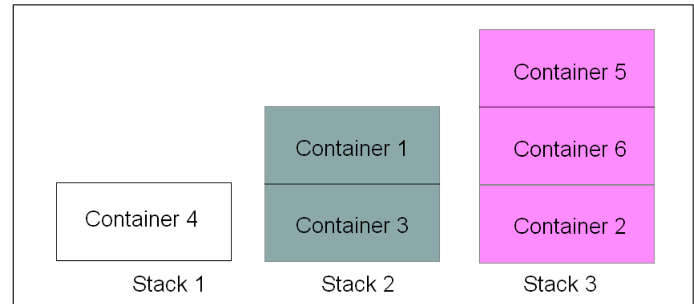


Figure 8. Stacking manner

If several containers are assigned to a same stack, they will be stored following the increasing order of their column numbers. This enables to take into account the arrival order constraint (5) and the departure times constraints (6) of the first mathematical model. For example, the containers 3 and 1 are both assigned to the stack 2, but the container 3 has the lowest storage location.

B. Algorithm

In the ant colony algorithm, which we propose to solve the container storage problem (ANTCSP), we use four parameters, which are: the number of ants (NA), the number of iterations (NT_{Max}), the minimum threshold of pheromone (τ_{Min}), and the maximum threshold of pheromone (τ_{Max}). This is based on the **Min-Max** version of the ant colony algorithm, more informations about the different versions of ant colony algorithms are available in [23]. The ant colony algorithm progresses as follows:

1: Initialization of pheromone.
2: Construction of a solution by each ant.
3: Evaluation of the solutions.
4: Initialization of the number of iterations, $NT = 1$.
5: While ($NT < NT_{Max}$) do:
    5.a: Update pheromone.
    5.b: Construction of new solutions by the ants.
    5.c: Evaluation of the solutions.
    5.d: $NT = NT + 1$.
  End while.

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

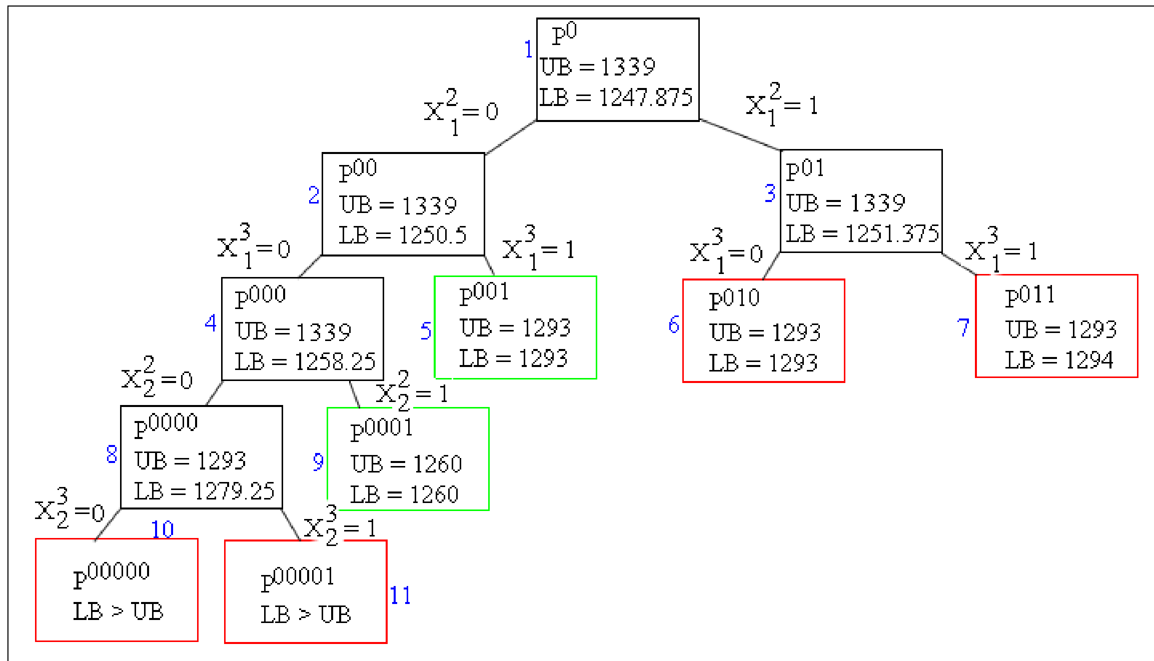


Figure 6. Search tree.

1) Construction method of a solution by an ant: The construction of a solution is done sequentially by adding successively elements. Therefore, before beginning the search of solutions, we firstly construct the set of options. This means, the set of couples (container: k, stack: p) that are compatible. In other words, every pair (k, p) that satisfies these three conditions:

- $c_p > 0$, (the stack p is not full)
- $r_p = R_k$, (the container k and the stack p have similar dimensions)
- $t_p \geq T_k$, (the departure time of the container k is inferior to that of the container that is already at the top of the stack p at the beginning of the new storage period).

All these couples form the set of options (for commodity, we name it E), and each option (k, p) has a pheromone trail, which is initialized $(\tau(k, p) = \tau_{Max})$.

In the step 2 of the ant colony algorithm, we use this following pseudo-code to construct a solution.

1: Let $S = \emptyset$, the solution that is being built.
2: $E_1 = E$.
3: Choose arbitrarily an element of E_1 and add it to S.
4: While (E_1 is not empty) do:
 4.a: Update E_1.
 4.b: Calculate the probability $P_{(k,p)}$ of every element (k, p) remaining in E_1.

$$P_{(k,p)} = \frac{(\tau_{(k,p)})^{\alpha} \times (\frac{1}{d_p^k})^{\beta}}{\sum_{(k,p) \in E_1} (\tau_{(k,p)})^{\alpha} \times (\frac{1}{d_p^k})^{\beta}}$$

 Where α and β are positive real numbers inferior to 1, and $\frac{1}{d_p^k}$ is the visibility.
 4.c: Choose the element of E_1 that has the largest probability and add it to S.
 End while.
5: If (the number of couples belonging to S is inferior to N) then
 5.a: Go back to Step 1.

Whenever a couple is added to the solution S, we remove it from the set of options E_1. After that, we decrease the capacity of the corresponding stack. And then, we delete from the set of options every couple that may compromise the validity of the solution. For example, suppose that we add to S the option (k, p). So, we update the capacity of the stack p (this means $c_p = c_p - 1$), and we remove from E_1 all couple (k', p') that satisfies at least one of these four conditions:

- $c_{p'} = 0$, (the stack is full)
- $k' = k$, (the container is already assigned)
- $O_k > O_{k'}$, (incompatible unloading numbers)
- $T_k < T_{k'}$, (incompatible departure times).

2) Method to update the pheromone trails: At the end of each iteration, the pheromone trails are updated in two steps. Firstly, an evaporation decreases the pheromone of each option, like follows:

$$\forall\ (k, p) \in E, \quad \tau_{(k,p)} = (1 - \rho)\tau_{(k,p)}$$

Where ρ is the evaporation rate, and $0 < \rho < 1$.
If $\tau_{(k,p)} < \tau_{Min}$, we adjust it $(\tau_{(k,p)} = \tau_{Min})$.

Unlike the evaporation, the augmentation of the pheromone trails is done only on the couples that belong to the best solution found during the current iteration. Let S_{bc} that solution, the pheromone trails of its couples are increased as follows:

$$\forall\ (k, p) \in S_{bc}, \quad \tau_{(k,p)} = \tau_{(k,p)} + \frac{1}{|O_{bc} - O_b + 1|}$$

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Where O_b is the value of the best solution found since the beginning of the algorithm until the current iteration, and O_{bc} is the value of S_{bc}. If $\tau_{(k,p)} > \tau_{Max}$, we adjust it ($\tau_{(k,p)} = \tau_{Max}$).

VIII. HYBRID ANT COLONY AND BRANCH-AND-CUT ALGORITHM

In the hybrid ant colony and branch-and-cut algorithm (HACBC), we use the ant colony algorithm to find an upper bound (**UB**) of the container storage problem. This upper bound is then used in the branch-and-cut algorithm, in order to accelerate it by only exploring the nodes that have lower bounds inferior to **UB**. The Hybrid algorithm is represented in Figure 9, where P^0 represents the initial integer problem and S designates the current best integer solution. The starter step and the final step are coloured in pink.

As can be seen in Figure 9, the HACBC algorithm is stopped only if the list of active nodes becomes empty. After the initialization of **UB** and S, the search tree is initialized with P^0, and then, these following actions are repeated in the same order until the stop condition:

- Select an element P^j in T.
- Relax and solve P^j with CPLEX.
- While there are violated valid inequalities, add them to the relaxation of P^j, and then, solve it again.
- If the solution of P^j is integer and better than S, update **UB** and S, remove from T every node that has a lower bound superior or equal to **UB**.
- If the solution of P^j is fractional and has a value inferior to **UB**, use the most fractional variable to perform a branching.
- Remove P^j from T.

IX. NUMERICAL RESULTS

In this section, we present the numerical results of the different algorithms that are proposed in this paper. The experiments were performed using a computer DELL PRECISION T3500 Intel Xeon 5 GHz processor. Each algorithm is implemented in C++ language. In addition, we use the CPLEX solver version 12.5, and the framework SCIP that is very useful because it allows a total control over the different components of the branch-and-cut algorithm.

The details concerning the data used in the numerical simulations are noted in Table III.

TABLE III. Benchmark set-up

Number of containers	$50 \leq N \leq 1400$
Number of stacks	$100 \leq N_p \leq 3500$
Maximum height of a stack	3
Percentage of vacant storage locations	$50\% \leq \frac{100 \times \sum_{p=1}^{N_p} c_p}{3 \times N_p}$
Number of sizes of containers	3 sizes: 20 feet, 40 feet, and 45 feet
Average dwell time of a container	4 days
Distance between the stack p and the quay where is the container k	$300\ m \leq d_p^k \leq 800\ m$

Before comparing the performances of the algorithms, we firstly researched the best values of the ant colony algorithm's parameters. To do this, we treat them individually. At each step, we vary the value of one parameter, the values of the other parameters do not change during the iterations. We applied this method on different instances, for each parameter, and then, we obtain the results described in Table IV.

To look for the suitable number of iterations (NT_{Max}), we considered the integers that are between 20 and 100, and we choose the number, from which the objective function does not decrease any more. Similarly, the number of ants (NA) are searched between 10 and 100. As for the exponent of the pheromone (α), the exponent of the visibility (β), and the rate of pheromone evaporation (ρ), they are dealt with by considering de real numbers that are between 0 and 1. And finally, the minimum threshold of pheromone (τ_{Min}) is sought by considering the integers that are between 1 and 5, while the maximum threshold of pheromone (τ_{Max}) is looked for by considering the integers that are between 5 and 10.

TABLE IV. Values of the ant colony algorithm's parameters.

Parameter	Value
Number of iterations	$NT_{Max} = 40$
Number of ants	$NA = 17$
Exponent of the pheromone	$\alpha = 0.3$
Exponent of the visibility	$\beta = 0.2$
Rate of pheromone evaporation	$\rho = 0.2$
Minimum threshold of pheromone	$\tau_{Min} = 1$
Maximum threshold of pheromone	$\tau_{Max} = 10$

Unlike the ant colony algorithm, the branch-and-cut algorithm and the hybrid algorithm give optimal results. However, the ant colony algorithm gives good upper bounds as can be seen in Table V, where the values of the objective function are mentioned for twenty four instances.

gap is the percentage of deviation, it is calculated using the following formula:

$$gap = \frac{Obj_{(ANTCSP)} - Obj_{(optimal)}}{Obj_{(optimal)}} \times 100$$

$Obj_{(ANTCSP)}$ is the value of the solution found by the ant colony algorithm, and $Obj_{(optimal)}$ is the value of the optimal solution found by the CPLEX solver. For the instances that could not be solved by CPLEX, $Obj_{(optimal)}$ represents the value of the optimal solution found by HACBC.

val is the value of the objective function.

The symbol $\cdots$ means that the execution is interrupted because it lasted more than 3 hours. Similarly, the symbol — means that the computer memory is insufficient to enable the resolution of the instance.

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

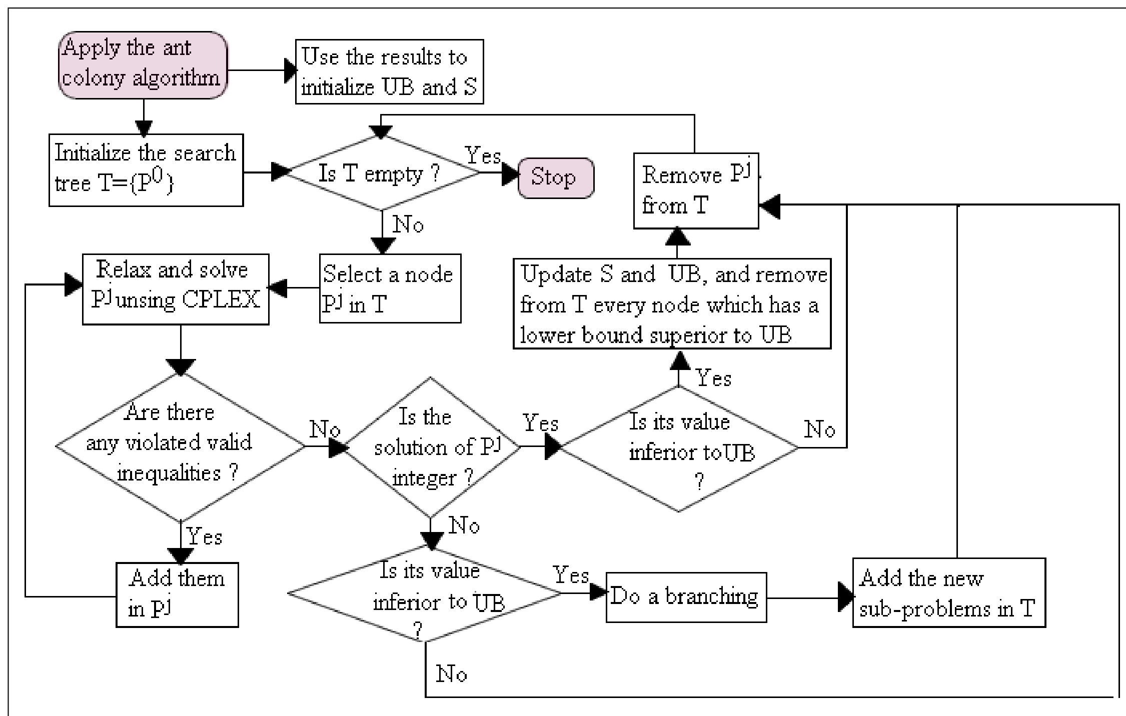


Figure 9. HACBC.

TABLE V. Comparison of the algorithms' solutions

N	N_p	ANTCSP		CPLEX	BC-CSP	HACBC
		val	*gap*	*val*	*val*	*val*
100	500	53682	0.85%	53230	53230	53230
150	500	62042	3.09%	60483	60483	60483
100	700	46672	4.02%	44867	44867	44867
100	1500	53945	1.81%	52988	52988	52988
50	200	15882	0%	15882	15882	15882
200	200	71682	1.43%	70671	70671	70671
80	100	25608	0%	25608	25608	25608
90	100	34636	0%	34636	34636	34636
100	100	39026	4.5%	37168	37168	37168
150	200	55388	0.69%	55011	55011	55011
100	3500	33035	1.5%	—	32547	32547
200	3500	67604	1.97%	—	66300	66300
300	3500	92406	1.57%	—	90979	90979
400	3500	125145	1.38%	—	123438	123438
500	3500	171137	1.33%	—	168895	168895
600	3500	184924	0.82%	—	183415	183415
700	3500	216424	0.6%	—	215138	215138
800	3500	245357	0.59%	—	243917	243917
900	3500	276831	0.95%	—	274238	274238
1000	3500	323187	0.87%	—	320415	320415
1100	3500	338955	0.48%	—	337347	337347
1200	3500	375269	0.47%	—	373498	373498
1300	3500	405622	0.64%	—	403054	403054
1400	3500	432816	0.39%	—	...	431116

The results depicted in Table V show that, in some cases, the ant colony algorithm gives optimal results. In addition, it gives generally good results that have percentages of deviation inferior to 5%.

In Table VI, we compare the execution times of CPLEX, BC-CSP, and HACBC. The ant colony algorithm is fast but as it does not give optimal results every time, it would not be relevant to compare its execution times to those of the other algorithms.

TABLE VI. Comparison of the execution times

N	N_p	HACBC	BC-CSP	CPLEX
100	500	0 sec	0 sec	2 min 58 sec
150	500	0 sec	1 sec	15 min 45 sec
100	700	0 sec	0 sec	4 min 11 sec
100	1500	0 sec	0 sec	11 min 16 sec
50	200	0 sec	0 sec	2 sec
200	200	1 sec	3 sec	14 min
80	100	0 sec	0 sec	1 min 5 sec
90	100	0 sec	0 sec	1 min 29 sec
100	100	0 sec	0 sec	2 min 11 sec
150	200	0 sec	1 sec	53 min 50 sec
100	3500	0 sec	0 sec	—
200	3500	0 sec	3 sec	—
300	3500	6 sec	14 sec	—
400	3500	11 sec	41 sec	—
500	3500	35 sec	1 min 36 sec	—
600	3500	2 min 35 sec	6 min 13 sec	—
700	3500	1 min 46 sec	5 min 57 sec	—
800	3500	3 min 20 sec	9 min 49 sec	—
900	3500	6 min 14 sec	15 min 35 sec	—
1000	3500	33 min 40 sec	1 h 41 min 26 sec	—
1100	3500	34 min 5 sec	1 h 43 min 47 sec	—
1200	3500	51 min 12 sec	2 h 5 min 4 sec	—
1300	3500	51 min 33 sec	2 h 21 min 20 sec	—
1400	3500	1 h 3 sec	...	—

As can be seen in Table VI, our branch-and-cut algorithm is quicker than the CPLEX solver version 12.5 for the resolution of the container storage problem. However, these two methods are outperformed by the hybrid ant colony and branch-and-cut algorithm.

The results that are contained in Table V and Table VI are obtained by doing a single execution for each instance.

X. Conclusion and Future Work

This paper deals with the inbound container storage problem at seaport terminal. A container terminal that uses straddle

carriers as handling and transfer equipments is considered. A mathematical model, which determines an accurate storage location for each container is proposed. This mathematical model takes into account physical and operational constraints, like the order, in which the containers are unloaded from vessels, and minimizes the total distance travelled by the straddle carriers between the quays and the container yard, in order to shorten the berthing times of the ships. A demonstration of the NP-hardness of the container storage problem is given. For the numerical resolution, we propose an efficient hybrid ant colony and branch-and-cut algorithm. This hybridization allows to improve the performances of the branch-and-cut algorithm that was proposed in [1]. In addition, it is an exact resolution method and is able to solve quickly large instances, which cannot be solved by the CPLEX solver.

In the future, we plan to test other branching rules and other valid inequalities. We also plan to study the case of container terminals that use automatic equipments like automatic guided vehicles and rail mounted cranes, and to propose other efficient resolution methods.

REFERENCES

[1] N. F. Ndiaye, A. Yassine, and I. Diarrassouba, "A Branch-and-Cut Algorithm to Solve the Container Storage Problem," in Proceedings of the ninth international conference on systems (ICONS), February 23–27, 2014, Nice, France, 2014, ISBN: 978-1-61208-319-3, ISSN: 2308-4243, URL: http://www.thinkmind.org/ [accessed: 2014-07-27].

[2] C. Zhang, J. Liu, Y. W. Wan, K. G. Murty, and R. J. Linn, "Storage space allocation in container terminals," Transportation Research Part B: Methodological, vol. 37, 2003, pp. 883–903.

[3] M. Bazzazi, N. Safaei, and N. Javadian, "A genetic algorithm to solve the storage space allocation problem in a container terminal," Computers & Industrial Engineering, vol. 56, 2009, pp. 44–52.

[4] C. Park and J. Seo, "Mathematical modeling and solving procedure of the planar storage location assignment problem," Computers & Industrial Engineering, vol. 57, 2009, pp. 1062–1071.

[5] D.-H. Lee, X. J. Cao, Q. Shi, and J. H. Chen, "A heuristic algorithm for yard truck scheduling and storage allocation problems," Transportation Research Part E: Logistics and Transportation Review, vol. 45, 2009, pp. 810–820.

[6] E. Kozan and P. Preston, "Mathematical modeling of container transfers and storage locations at seaport terminals," OR Spectrum, vol. 28, 2006, pp. 519–537.

[7] S. Sauri and E. Martin, "Space allocating strategies for improving import yard performance at marine terminals," Transportation Research Part E: Logistics and Transportation Review, vol. 47, 2011, pp. 1038–1057.

[8] K. H. Kim and H. B. Kim, "Segregating space allocation models for container inventories in port container terminals," International Journal of Production Economics, vol. 59, 1999, pp. 415–423.

[9] C. Jinxin, S. Qixin, and D.-H. Lee, "A Decision Support Method for Truck Scheduling and Storage Allocation Problem at Container," Tsinghua Science & Technology, vol. 13, 2008, pp. 211–216.

[10] M. Yu and X. Qi, "Storage space allocation models for inbound containers in an automatic container terminal," European Journal of Operational Research, vol. 226, 2013, pp. 32–45.

[11] R. Moussi, N. F. Ndiaye, and A. Yassine, "Hybrid Genetic Simulated Annealing Algorithm (HGSAA) to Solve Storage Container Problem in Port," Intelligent Information and Database Systems, Lecture Notes in Computer Science, vol. 7197, 2012, pp. 301–310.

[12] K. H. Kim and K. Y. Kim, "Optimal price schedules for storage of inbound containers," Transportation Research Part B: Methodological, vol. 41, 2007, pp. 892–905.

[13] P. Preston and E. Kozan, "An approach to determine storage locations of containers at seaport terminals," Computers & Operations Research, vol. 28, 2001, p. 983995.

[14] K. H. Kim, Y. M. Park, and K.-R. Ryu, "Deriving decision rules to locate export containers in container yards," European Journal of Operational Research, vol. 124, 2000, pp. 89–101.

[15] L. Chen and Z. Lu, "The storage location assignment problem for outbound containers in a maritime terminal," International Journal of Production Economics, vol. 135, 2012, pp. 73–80.

[16] Y. J. Woo and K. H. Kim, "Estimating the space requirement for outbound container inventories in port container terminals," International Journal of Production Economics, vol. 133, 2011, pp. 293–301.

[17] K. H. Kim and K. T. Park, "A note on a dynamic space-allocation method for outbound containers," European Journal of Operational Research, vol. 148, 2003, p. 92101.

[18] N. Nishimura, A. Imai, G. K. Janssens, and S. Papadimitriou, "Container storage and transshipment marine terminals," Transportation Research Part E: Logistics and Transportation Review, vol. 45, 2009, p. 771786.

[19] B. Dushnik and E. W. Miller, "Partially ordered sets," American Journal of Mathematics, vol. 63, 1941, pp. 600–610.

[20] K. Jansen, "The mutual exclusion scheduling problem for permutation and comparability graphs," STACS 98, Lecture Notes in Computer Science, vol. 1373, 1998, pp. 287–297.

[21] F. Bonomo, S. Mattia, and G. Oriolo, "Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem," Theoretical Computer Science, vol. 412, 2011, pp. 6261–6268.

[22] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization by a colony of cooperating agents," Transactions on Systems, Man, and Cybernetics-Part B, IEEE, vol. 26, 1996, pp. 1–13.

[23] M. Dorigo and T. Stützle, "The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances," Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, 2003, pp. 250–285.

2014, © Copyright by authors, Published under agreement with IARIA - www.iaria.org