An experimental comparison of different metaheuristics for the Master Bay Plan Problem[1]

Daniela Ambrosino [a], Davide Anghinolfi [b], Massimo Paolucci [b,], Anna Sciomachen [a*]

a Department of Economics and Quantitative Methods (DIEM), University of Genoa
Via Vivaldi 5, 16126 Genova, Italy
{ambrosin,sciomach}-@economia.unige.it,
b Department of Communication, Computer and Systems Sciences (DIST), University of Genoa
Via Opera Pia 13, 16145 Genova, Italy
paolucci@dist.unige.it, davide.anghinolfi@unige.it

Abstract. Different metaheuristics for the problem of determining stowage plans for containerships, that is the so called Master Bay Plan Problem (MBPP) are compared.
The first approach is a tabu search (TS) metaheuristic and it has been recently presented in literature. Two new solution procedures are proposed in this paper: a fast simple constructive loading heuristic (LH) and an ant colony optimization (ACO) algorithm.
An extensive computational experimentation performed on both random instances and real size ones is reported and conclusions on the appropriateness of the tested approaches for the MBPP are drawn.

Key words: metaheuristics, stowage plan

1. Introduction and problem definition

The stowage of the containers on the ship, that is the so called master bay plan problem (MBPP), is daily faced by each terminal management. This problem can be defined as follows: given a set C of n containers of different types to load on the ship and a set S of m available locations within the ship, we have to determine the assignment of the containers to the ship locations in order to satisfy the given structural and operational constraints related to both the ship and the containers and to minimise the total loading time.

Each location is addressed by three indices i, j, k representing, respectively: the bay (i), that gives the position of the location related to the cross section of the ship (counted from bow to stern); the row (j), that gives the position related to the horizontal section of the corresponding bay (counted from centre to outside); the tier (k) that gives the position related to the vertical section of the corresponding bay (counted from bottom to top). Let I, J and K be, respectively, the set of bays, rows and tiers of the ship.

Note that according to a practice adopted by the majority of maritime companies, bays with even number are used for stowing 40' containers and correspond to two contiguous odd bays that are used for 20' containers. Thus, we denote with $E \subset I$ the subset of even bay and with $O \subset I$ the one of odd bays. Set C is partitioned into two subsets, T and F, consisting, respectively, of 20 and 40 feet (40') containers.

The sets of tiers in hold and on deck are respectively denoted as K^H and K^D. The locations identified by the same pair of bay and row (i, j), $i \in I$, $j \in J$, correspond to a *stack*; in general a pair (i, j) is associated both with a stack in the hold, defined

[1] ***This work has been developed within the research project*** *"Container import and export flow in terminal ports: decisional problems and efficiency analysis" PRIN 2007j494p3_005, Italy*

[*] Corresponding author: sciomach@economia.unige.it; tel: +39 010 2095484; fax: +39 010 2095243

as $P_{i,j}^{H} = \{(i, j, k) : k \in K^{H}\}$, and a stack on deck, denoted as $P_{i,j}^{D}$ and similarly defined. Let P be the set of all the available stacks for a ship.

We assume that the container handling operations are performed by quay cranes, which are positioned on the quay side of the ship. Then, we denote with $L \subset J$ the subset of rows located from the middle of the ship to the quay side, whereas with $R \subset J$ the subset of rows nearest to the sea side for which loading operations are more time consuming.

The considered MBPP involves only the loading decisions at the first port without taking into account the possible loading operations at the next ports in the ship route. This kind of problem is usually faced by the terminal manager. In [12], the problem of defining stowage plans in spitted into a two-step process concerning respectively the shipping line and the terminal manager, and the authors provide for each step a review of the corresponding optimization models. A relevant literature update is provided in [11].

In this work we also assume that the ship is empty and only one yard crane is available. Despite of such simplifying assumptions, the resulting mathematical formulation is not easy. In fact, in order to model the MBPP we must deal with some basic combinatorial optimization constraints as assignment constraints (i.e., each container must be assigned at most to a single location and each location must receive at most a single container) and knapsack constraint (the total weight of the containers loaded on the ship cannot be greater than the total ship capacity); furthermore, some constraints related to the size, weight and destination of the containers, and others relevant to the ship stability, have to be taken into account. Stability conditions involve three different types of equilibrium: the *horizontal equilibrium* condition imposing that the weight of the left size of the ship (quay side) should be equal (within a given tolerance) to the weight of the right side (sea side); the *cross equilibrium* condition, necessary to avoid bending ($\cap$) or saddling ($\cup$) of the ship, imposing that the weight of the anterior part of the ship (centre-bow) should be equal (within a given tolerance) to the weight of the posterior part of the ship (centre-stern); the *vertical equilibrium* condition imposing that the weight of each tier must be greater than or equal to the weight of the tier immediately over it. Destination constraints state that containers having as destination the last port of the ship route must be loaded first, i.e. into lower tiers; analogously, containers bound for the first port in the route must be loaded last.

MBPP is a NP-Hard [5]. Some Integer Programming models for the MBPP have been proposed in [6] and in [10]; unfortunately these papers deal with simplified version of problem so that the proposed models are not suitable for real life large scale applications. In [1] the details of the MBPP are provided and a 0/1 Integer Programming (IP) model is presented. In that model decision variables are related to the assignment of each container to a location of the ship; the model has been used for solving up to optimality only very small instances. Successively, in [3] a new 0/1 IP model is proposed where variables correspond to the assignment of ship locations to groups of containers, each characterized by a range of weight (e.g., low, medium or high), a specific type and a destination; using that model, due to the smaller number of variables and constraints, the authors have been able to solve larger instances. However, even considering groups of equivalent containers not always it has been possible to find a integer feasible solutions within a reasonable CPU time, that is some hours. This last 0/1 IP model works on groups of containers, can account for the presence of hatches between hold and deck locations, and allows the load on board of dangerous goods usually pre-stowaged near hatches.

In [3] the MBPP is solved by using a *bay assignment* procedure to assign the subsets of containers with a single destination to different ship partitions, corresponding to subsets of bays. Then, we solved a single destination 0/1 IP model where also the ship stability weight constraints were relaxed. Finally, possible infeasibilities of the global solution due to the violation of cross and horizontal stability conditions were removed by means of a tabu

search procedure. The tabu search algorithm is also adopted to improve the global solution, i.e., to reduce the total loading time. It is based on seven classes of moves which combine three kinds of items that can be moved, i.e., a single container, a stack of containers and a bay, with three kinds of position exchanges, i.e., anterior-posterior location exchange, left side-right side exchange, and cross exchange.

With this heuristic method we are able to check and force feasibility up to small-medium sized instances. Anyway, for larger instances, it is not possible to use the 0/1 IP model for obtaining an initial solution. For this reason we here propose a simple constructive procedure. Moreover, in this paper we describe a new metaheuristic approach that we apply successfully to very large instances. The new approach is an Ant Colony Optimization (ACO) metaheuristic.

The remaining of the paper is organized as follows. In Section 2 the ACO algorithm is described, while in Section 3 a simple heuristic for generating feasible solutions for MBPP is given. Section 4 reports the performed experimental analysis based on different classes of instances; finally, conclusions are drawn in Section 5.

2. The proposed ACO approach for the MBPP

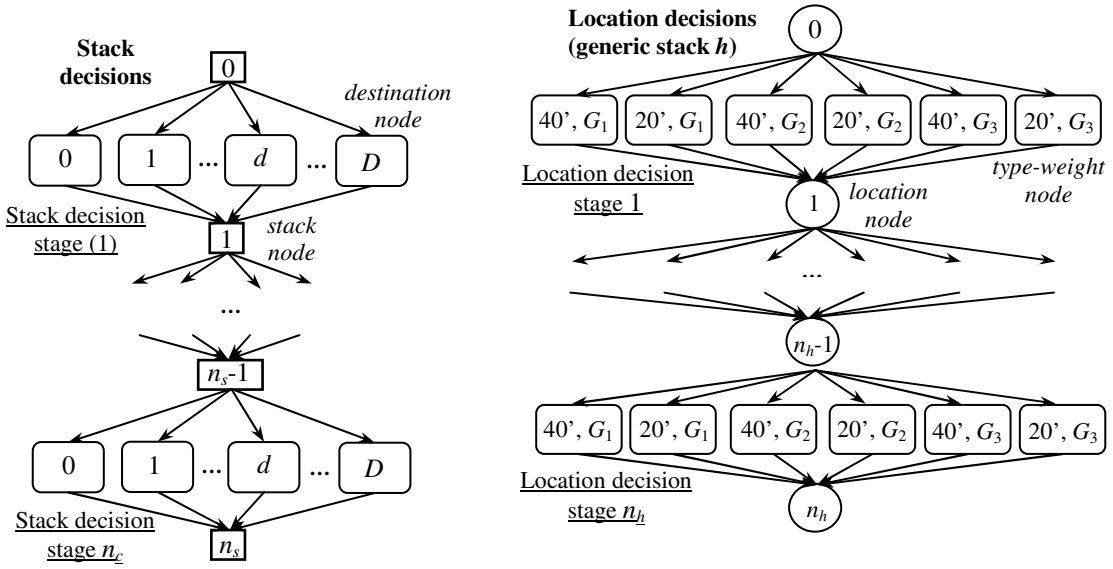
In this section we describe the main aspects of an ant colony optimization (ACO) approach for the MBPP. ACO is a population-based metaheuristic which tries to emulate the successful behaviour of real ants in cooperating to find shortest paths to food for solving combinatorial problems ([7],[9]). Real ants have an effective indirect way to communicate each other which is the most promising trail towards food: ants produce a natural essence, called pheromone, which they leave on the followed trail to food in order to mark it. The pheromone trail evaporates with time and it disappears on the paths abandoned by the ants, but differently it can be reinforced by the passage of further ants: thus, effective (i.e., shortest) paths leading to food are finally characterized by a strong pheromone track, and they are followed by most of ants. The ACO metaheuristic has been the subject of both theoretical studies and successfully applied to many combinatorial optimization problems.

We consider a set of m artificial ants. At each iteration the ants progressively fix the destination, type and class of weight for the available ship locations, and thus they construct a solution by accordingly assigning the containers to the locations. After that, a LS is executed starting from the best solution found in the current iteration, a global pheromone update phase takes place and the whole process is iterated. The algorithm terminates when a maximum number of iterations is reached.

The proposed ACO is the adaptation of the algorithm introduced in [4]. Such an approach is inspired by the Ant Colony System (ACS) [8] and *Max-Min* Ant System (MMAS) [13] versions of the basic algorithm and it includes a new local and global pheromone update mechanism. For the sake of brevity, hereinafter we focus only on the modelling aspects highlighting how the ACO approach has been applied to the MBPP, since the details of the algorithm can be found in [4]. On the other hand, readers interested in a comprehensive presentation of the ACO metaheuristic can find a valuable and general reference in [7].

Let $CD=\{1,...,D\}$ denotes the set of destinations for the containers in C ordered according to the ship route. At each iteration the ants take a sequence of decisions nested in two levels: *stack decision level* and *location decision level*. At the higher stack decision level the ants consider a stack at a time and fix the latest destination for the containers loaded in its locations (e.g., if d^{max} is the latest destination for a stack, then only containers bound for $1,...,d^{max}$ can be loaded in it). At the lower location decision level, the ants establish the type $t \in \{20', 40'\}$ and class of weight $g \in G$ for each location in the considered stack, finally

assigning to it a container with a compatible destination, type and class of weight, which satisfy both destination and vertical equilibrium constraints.
Such decisions correspond to the search of a path from a start node (the ant colony *nest*) to an end node (the *food*) in a *construction graph* structured in two nested levels, as depicted in Figure 1.



(1.a) The stack decision level

(1.b) The location decision level

Fig. 1: The construction graph of the ACO algorithm

Figure 1.a shows the graph corresponding to the stack decision level; here the ants determine a path from the stack node 0 (the ant nest) to the stack node n_s (associated with the last considered stack), selecting at each stage one destination in $CD \cup \{0\}$, where 0 denotes that the stack is not assigned. The order according to which the stacks are considered in the stack decision stages is heuristically fixed for both favouring the ship stability and reducing the loading time. In particular, we built a sorted list of rows in increasing order of the associated loading time t_j, $j \in J$ (computed as $t_j = \max_{k \in K} t_{jk}$, being t_{jk}, $j \in J$, $k \in K$, the loading time for row j and tier k) but alternating a left row and a right one. Similarly, we built a sorted list of bays alternating a bay in the middle of the ship, one in the anterior part and then one in the posterior part with a procedure similar to the pre-assignment performed in [2]. Finally, the sequence of stacks defining the stack decision stages is produced by extracting the index i and the index j respectively from the sorted lists of bays and rows.
After a stack level decision the ants proceed considering the locations in the stack. Figure 1.b details the location decision level for a generic stack h. Also in this case the ants' decisions correspond to find a path from the location node 0 to the location node n_h (associated with the last available location in stack h), determining at each stage the type and class of weight (here $G=\{G_1, G_2, G_3\}$, where G_1 stands for light, G_2 for medium and G_3 for heavy) for the location so no vertical equilibrium constraint or type compatibility is violated. In the following we denote with N_s, N_d, N_l and N_{tw} respectively the sets of stack, destination, locations and type-weight nodes. Note that (a) in the stack decision level of the construction graph the stage associated with a stack on deck immediately follows the corresponding stack in hold; (b) in the location decision level the location nodes are ordered from the lowest to

the higher. After having fixed at a location decision stage both the latest destination and the type and class of weight for a location, the ants actually assign to it a specific compatible container selected from the not yet loaded ones which that do not violate any destination constraint (this is simply obtained sorting the available compatible containers in decreasing order of destination).

Note that n_s may be greater than the number of available stacks on the ship since the stacks with available locations may be reconsidered whenever there are not assigned containers. We must remark that the number of the stages considered at the location decision level may vary since the ants' decisions at a stage constraint the available alternatives at the successive stages: in particular, a single type of container is allowed for a stack, and weight and destination constraints can prohibit certain assignment patterns (for example, the latest destination for containers in a stack $P_{i,j}^{D}$ on the deck must be not greater than the earliest destination of containers in the correspondent stack $P_{i,j}^{H}$ in the hold. Therefore, assuming $n_s=|P|$, we have $2\cdot|I|\cdot|J|$ stack decision stages and at most $|K^H|$ and $|K^D|$ location decision stages for the locations associated with a stack respectively in the hold and on the deck, which are in turn composed by $3\cdot|K^H|$ and $3\cdot|K^D|$ pairs of type-weight nodes.

The information composing the ants state are (a) the sets of containers that remain to be located $C_R=\{(p,\ t,\ g,\ nc_{ptg})\colon\ p\in CD,\ t\in\{20',\ 40'\},\ g\in G\}$, where nc_{pgt} is the number of containers to be loaded with destination p, type t and group of weight g; (b) the partial set of decisions for stow locations $C_A=\{(i, j, k, p, t, g)\colon i\in I, j\in J, k\in K, p\in D, t\in\{20', 40'\}, g\in G\}$.

A pheromone trail is associated with a subset U of the arcs of the construction graph: U includes the arcs (s, d): $s\in N_s$, $d\in N_d$ connecting stack nodes to destination nodes at the stack decision level, and the arcs (l, z): $l\in N_l$, $z\in N_{tw}$ connecting location nodes to type-weigh nodes at the location decision level.

The ant solution construction process outlined so far is quite flexible as it allow to locate containers with different destinations in a same stack or bay. The selection of a destination node and type-weigh node at each stage of the two decision levels is performed in two steps similarly to the ACS: first an ant determines the node selection rule between *exploitation* and *exploration*, then it actually select the node. The ant extracts a random number q~[0,1] from the uniform distribution and chooses exploitation if $q\le q_0$ (where $q_0\in[0,1]$ is a fixed parameter), otherwise exploration. The *exploitation* rule at a generic decision node s such that $(s, h)\in U$ selects the next node h^* in a deterministic way as

$$h^* = \arg\max_{h:(s,h)\in U}\{\tau_e(s,h)\cdot[\eta(s,h)]^\beta\} \tag{1}$$

whereas the *exploration* rule according to a *selection probability* $\pi_e(s,h)$ computed as

$$\pi_e(s,h)=\frac{\tau_e(s,h)\cdot[\eta(s,h)]^\beta}{\sum_{z:(s,z)\in U}\tau_e(s,z)\cdot[\eta(s,z)]^\beta} \tag{2}$$

The subset of arcs $(s, h)\in U$ identifies the solution components and $\tau_e(s,h)$ is the pheromone trail associated with the component (s, h) at iteration e. The pheromone trails represent the ACO learning device and provide a measure of the appropriateness of selecting a component during the construction of "good" solutions. Following the same approach in [4], the pheromone values assigned to $\tau_e(s,h)$ are independent of the objective function values associated with previously explored solutions including the component (s, h); pheromone values vary in an arbitrary range $[\tau_{Min},\tau_{Max}]$, with $\tau_{Min}<\tau_{Max}$, which is fixed

independently of the specific problem or instance considered (actually the algorithm behaviour does not depend on the choice of τ_{Max} and τ_{Min}). The pheromone trails are initialized as $\tau_0(s,h) = (\tau_{Max} + \tau_{Min})/2$ and their variation in the range $[\tau_{Min}, \tau_{Max}]$ during the exploration process, i.e., the ant colony learning mechanism, is controlled by a same global pheromone update rule proposed in [4] that allows a smooth variation of $\tau_e(s,h)$ within these bounds such that both extremes are asymptotically reached. The quantity $\eta(s,h)$, associated with the component (*s*, *h*), is a heuristic value that is used to direct the ants' selections during the first iterations when the pheromone trails are almost the same for all the components. We based the computation of $\eta(s,h)$ on a very simple rule which return $\eta(s,h) = \eta_t(s,h) \cdot \eta_c(s,h)$, where $\eta_t(s,h)$ and $\eta_c(s,h)$ are the heuristic values relevant to the choice respectively of the type of containers (i.e., the use of an even bay or the two paired odd bays) and of the class of weight for the location associated with *h*. In order to minimize the loading time we favour the assignment of 20' containers to the locations in the rows closer to the berth side, fixing $\eta_{20}(s,h) = 2$ for the relevant nodes in the construction graph, as well as the assignment of the heaviest containers to the lowest tiers in the hold and on the deck, fixing $\eta_t(s,h) = 2$ for the relevant nodes, letting $\eta_t(s,h) = \eta_g(s,h) = 1$ in all the other cases. The quantity β in (1) and (2) is a parameter representing the importance of the heuristic value with respect to the pheromone trail in the ant selection rules.

Whenever an ant reaches the final state node n_s a tentative solution *x* to the MBPP is build. Tentative solutions could not be feasible as some equilibrium constraints could be violated and some containers could be not loaded. Then, we compute the global cost for the solution as

$$Z(x) = M_s(\sigma_1(x) + \sigma_2(x)) + M_{nl}\mu(x) + L(x) \tag{3}$$

being $\mu(x)$ the number of not loaded containers and M_s and M_{nl} the two penalties respectively for the violation of stability constraints and for not loaded containers in *x*. After all the ants *a*=1,..., *m* have determined a solution x_e^a at an iteration *e*, the best solution found in the iteration, i.e., $x_e^{best} = \arg\min_a Z(x_e^a)$, is determined and a LS step takes place. The purpose of this LS is that of perturbing x_e^{best} by means of a set of moves that change the locations of a subset of loaded containers in order to eliminate the possible violation of stability constraints and to improve the overall loading time. The LS procedure at each iteration performs the following sequence of three types of random moves, whose details are provided in [3]: Anterior-Posterior exchange of containers (APC); Left-Right side exchange of containers (LRC); Cross exchange of containers (CEC). As regards the solution feasibility, we should note that the LS is devoted only to recover violations of stability which, emerging from the solutions considered as a whole, are difficult to avoid during the ACO solution construction process. Thus, the objective function minimized by the LS disregards the $M_{nl}\mu(x)$ component since, leaving in this way the task of loading all the required containers to the ACO solution construction process. The LS adopts the first improvement move acceptance rule and it terminates after a fixed maximum number of iterations. However, whenever the overall ACO algorithm reaches a maximum number of non improving iterations, the LS maximum number of iterations is temporarily doubled until an improved solution is found.

Finally, the pheromone trails associated with the solution components included in the best solution found so far are reinforced while the other components are evaporated according to the global pheromone update rule introduced in [4], and the algorithm iterates.

3 A simple constructive heuristic

In this section we describe a new solution method consisting of a simple constructive loading heuristic (LH).

We will compare the solutions produced by LH with the ones obtained by the other proposed solution methods. Moreover, when facing very large and real size instances, we use LH to generate a starting solution for the Tabu search proposed in [3].

The LH determines a solution that satisfies both the destination constraints and weight constraints as follows.

1. Let C'={ C_1 , C_2 ,… C_h,…. C_D} be the partition of set C of containers, and C_h be the set of containers having as destination port h. Split the containers in C_h in accordance with their type obtaining sets C_{hT} and C_{hF} respectively for 20' and 40' containers.
2. Apply the bay partitioning procedure to determine for each destination h the subset of bays $I_h \subseteq I$ assigned to destination h.
3. For each destination h, starting from the last one (D) to the first one (1), first assign the subset of 20' containers and then the subset of 40' containers as follows.
   - 3.1 Sort C_{hX} (where X denotes the generic container type) in increasing order of the container weights.
   - 3.2 Repeat,
     - 3.2.1 Select $i \in I_{hX}$ (a bay compatible with type X)
     - 3.2.2 For k=1 to K (tiers) and j=1 to J (rows) assign each $c \in C_{hX}$ to location (i, j, k) starting from the first container in the set (i.e., the heavier one), then set $C_{hX}=C_{hX}\backslash\{c\}$ and $I_{hX}=I_{hX}\backslash\{i\}$

     Until $C_{hX}\neq\emptyset$ or $I_{hX}=\emptyset$
   - 3.3 If $C_{hX}\neq\emptyset$ and $I_{hX}=\emptyset$ try to locate the remaining containers without violating destination and weight constraints as follows
     - 3.3.1 If $C_{hT}\neq\emptyset$, the remaining containers are possibly located above 20' containers with destination $h'>h$ without violating the weight constraints.
     - 3.3.2 If $C_{hF}\neq\emptyset$, the remaining containers are possibly located above 20' containers with the same destination, 40' containers with destination $h'> h$, or 20' containers with destination $h'> h$, in any case without violating the weight constraints.

4. Experimental results

The proposed MBPP algorithms were coded in C++, using the commercial Cplex 9.0 as 0/1 IP solver, and tested on a 1.5GHz, Intel Celeron PC with 1Gb RAM. The tests were related to two containerships of different size. The first containership is a real medium size one, the European Senator, whose data have been provided by the SECH Terminal of Genova, Italy. The European Senator has a 2124 TEU capacity and it is composed by 17 odd bays, 10 rows and 6 tiers in the hold and 21 odd bays, 12 rows and 5 tiers in the upper deck. We consider 14 medium size test instances that have a number of containers ranging from 715 to 1413 corresponding to TEUs ranging from 945 to 1800, with 2 or 3 destinations. Then we consider a larger containership with a capacity of 5632 TEUs, and we random generated 11 instances that have a number of containers ranging from 3800 to 4110 corresponding to TEUs ranging from 4920 to 5510, with 4 or 5 destinations.

Tables 1 and 2 report the characteristics of the above mentioned instances, showing the total number of containers, both in *TEU* and absolute number (*Number of containers*),

the percentage of 20' and 40' containers (*Type*), the percentage of containers for three groups of weight (*Weight*), the partition of containers for each destination (*Destination*) and finally

Table 1: The test instances for the medium size ship (*European Senator*).

Instance	TEU	Number of containers	Type (%)		Weight (%)			Destination (%)			Occupancy (%)
			20'	40'	G1	G2	G3	1	2	3	
1	945	715	68	32	30.1	60.0	9.9	48.99	51.01	-	52.50
2	1022	762	66	34	29.9	60.1	10.0	49.02	50.98	-	56.78
3	1120	820	63	37	30.0	60.0	10.0	49.02	50.98	-	62.22
4	1218	898	64	36	30.1	60.2	9.7	49.26	50.74	-	67.67
5	1380	1090	65	35	30.0	60.0	10.0	48.99	51.01	-	76.67
6	1386	984	73	27	30.1	60.0	10.0	49.06	50.94	-	77.00
7	1420	1060	59	41	30.0	60.0	10.0	49.01	50.99	-	78.89
8	1528	1202	68	32	30.0	60.0	10.0	49.02	50.98	-	84.89
9	1627	1215	66	34	30.0	60.0	10.0	49.42	50.58	-	90.39
10	1800	1413	73	27	30.0	60.0	10.0	49.00	51.00	-	100.00
11	1320	980	66	34	30.1	60.1	9.8	34.09	38.94	26.97	73.33
12	1415	1069	66	34	30.0	60.1	9.9	33.99	37.03	28.98	78.61
13	1522	1138	70	30	30.0	60.0	10.0	34.43	38.50	27.07	84.56
14	1724	1331	73	27	30.1	60.0	9.9	34.11	38.81	27.09	95.78

the level of occupancy of the ship (*Occupancy*).

Table 2: The test instances for the Large Containership

Instance	TEU	Number of containers	Type (%)		Weight (%)			Destination (%)					Occupancy (%)
			20'	40'	G1	G2	G3	1	2	3	4	5	
1	4920	3800	71	29	38.4	29.8	19.2	21.8	21.8	21.8	21.8	-	87.36
2	5000	3750	67	33	35.5	35.5	17.8	22.2	22.2	22.2	22.2	-	88.78
3	5080	3800	66	34	36.1	31.5	22.6	22.5	22.5	22.5	22.5	-	90.20
4	5370	4000	66	34	47.2	31.4	16.7	39.1	21.3	25.6	9.4	-	95.35
5	5437	4083	67	33	59.1	25.7	11.8	28.1	25.2	22.9	20.3	-	96.54
6	5505	4110	66	34	50.6	31.6	15.6	27.9	25.1	23.5	21.2	-	97.75
7	5100	4100	76	24	46.34	31.43	12.78	20.60	20.95	19.35	19.89	9.77	90.55
8	5400	4000	65	35	47.94	33.56	14.38	19.18	19.18	19.18	19.18	19.18	95.88
9	5004	3451	55	45	35.53	31.11	22.21	21.91	19.28	19.28	15.50	12.87	88.85
10	5004	3451	55	45	35.53	31.11	22.21	12.87	15.50	19.28	19.28	21.91	88.85
11	5510	3800	55	45	48.92	34.27	14.65	19.57	19.57	19.57	19.57	19.57	97.83

The loading times depend on the row and tier and grow from left side rows to the right ones and from highest tiers on the deck to the lowest ones in the hold; times are expressed in 1/100 of minute and range from 120 to 330.

We first tried to find a global optimal solution for the instances in Tables 1 and 2 using the exact 0/1 IP model given in Ambrosino et al. (2009). We fixed as termination conditions for the solver an absolute gap = $5 \cdot Nd$ minutes, where Nd is the number of destinations of the considered instance, and a maximum time limit of 1 hour, and we obtained the following results.

- Medium size ship (instances in Table 1):
  - no integer solution was found in one hour of computation for all the 4 instances with 3 destinations;
  - the solver stopped with a feasible solution after reaching the maximum time limit for 5 instances out of 10 with 2 destinations;
  - the solver required on the average after 16m and 47s to terminate for the remaining 5 instances with 2 destinations.
- Large size ship (instances in Table 2): no integer solution was found even extending the maximum computation time to two hours.

Note that finding an exact solution for the instances with more than two destinations was significantly hard; therefore the need of an effective heuristic for medium or large containerships is apparent.

We tested the TS and the ACO approaches comparing the obtained results also with the ones produced by the exact 0/1 IP model for medium size instances and by the simple constructive LH followed by the TS (LH-TS) for large size instances.

We performed some preliminary tests to select suitable values for the TS and the ACO parameters, identifying the following configurations: for the TS we fixed tenure = 80, diversification after 30 non improving iteration, diversification length = 35 iterations, termination conditions corresponding to maximum number of iterations = 500 and maximum not improving iterations = 50; for the ACO we used 80 ants, the pheromone evaporation factor α=0.05, q_0=0.95, β=1, maximum number of non improving iterations = 8 and maximum number of iterations =1000. Since random choices are used in the both in TS and ACO, five independent runs were performed for each instance and the average results were considered.

Table 3 shows the results for the medium size test instances. We report in the first group of columns (*Exact 0/1 IP*) the results produced by the exact 0/1 IP model, followed by three groups of columns showing respectively the starting solutions of the TS obtained by solving the 0/1 IP model for the single destination MBPP (*SD-MBPP*), the final solutions yielded by the TS (*Tabu Search*) and the final solutions produced by the ACO. An absolute gap of 5m was fixed as termination condition for the SD-MBPP 0/1 IP model and the loading times shown in the columns *Obj* are expressed as 1/100 of minute. Note that *Time (a)* and *Time (b)* columns report the CPU times in seconds needed by the 0/1 IP solver respectively for the exact model and the single destination model, whereas the *Avg Time* columns for the TS and ACO show the total CPU times needed by the two metaheuristic approaches. The last table row finally shows the overall average CPU times for this set of instances.

Table 3: The results for the medium size ship.

		Exact 0/1 IP (max time=1h, absolute gap=5×N_d m)		**SD-MBPP (absolute gap=5m)**				**Tabu Search (average over 5 runs)**		**ACO (average over 5 runs)**	
Ist.	**Nd**	**Obj (a)**	**Time (a)**	**Obj (b)**	**σ_1**	**σ_2**	**Time (b)**	**Avg obj**	**Avg Time**	**Avg obj**	**Avg Time**
1	2	139530	1300.8	150570	15	5380	68.4	142990	101.1	144162	142.0
2	2	149440	1606.4	155100	20	1065	11.6	151700	43.8	154918	158.6
3	2	161670	1876.7	168310	0	1225	7.4	164444	40.5	167780	180.1
4	2	178320	1613.3	186310	0	1435	21.8	180906	54.3	183336	174.0
5	2	219650	3698.6	225510	0	1115	17.6	222600	58.1	227486	189.2
6	2	197410	1999.8	203040	0	1555	18.0	200072	49.7	203566	191.6
7	2	213520	3698.5	219330	0	1235	12.1	216036	52.0	221572	154.9
8	2	244660	3698.4	252640	0	1495	21.3	247810	63.5	254168	141.8
9	2	248050	3295.0	253640	5	1010	212.3	250128	243.4	258578	137.9
10	2	293150	3704.7	293310	0	1960	17.0	293530	41.5	305228	115.7
11	3	-	3600.0	206590	20	5685	65.0	200140	119.6	204452	168.2
12	3	-	3600.0	220210	15	1300	7.5	217660	45.5	224822	181.2
13	3	-	3600.0	237050	0	1625	19.8	233608	65.9	239912	183.7
14	3	-	3600.0	278780	0	470	30.8	275964	66.7	285948	176.3
		Averages	2953.4				37.9		74.7		167.7

We must first remark that both the TS and the ACO approaches were able to find solutions satisfying both the cross and the horizontal stability conditions on every run for each instance. The average CPU times highlighted in Table 3 then underline that both the compared metaheuristics were able to provide feasible solutions in a fraction of the time needed by the exact 0/1 IP model. We compare the results of instances 1-10 in Table 3 by computing the percentage deviations of the solutions yield by the SD-MBPP model and the TS and ACO approaches from the ones found by the exact 0/1 IP model. The solutions obtained by the SD-MBPP model are worse but not very far from the ones yielded by the exact 0/1 IP model: the average percentage deviation of 3.41%, which corresponds to an average difference in the total loading time = 1h 2m 21s, was obtained with a very short average CPU time needed (37.9s); however, we must remark that the SD-MBPP was never able to satisfy the stability conditions (i.e., $\sigma_1,\sigma_2 > 0$), especially the horizontal one. The best results are apparently due to the TS approach that was only 1.33% worse (corresponding to an average difference in the total loading time = 24m 49s) than the exact 0/1 IP model that needed a very much longer computation. Compared to TS, the designed ACO algorithm presents worse performances with respect to average loading times and computation requirements.

For the instances with 3 destinations (11-14 of Table 3) we compared results obtained by TS and ACO, observing that the loading times produced by the ACO were worse on average of 2.94% than the ones obtained by the TS.

The results obtained for the large size instances are reported in Tables 4 and 5. When we consider the large instances with more than 3 destinations (as the ones of Table 2), the ACO algorithm turned out to be the best one, and in some cases it was the only approach able to generate a solution.

Note that for these instances the exact 0/1LP model was never able to determine a feasible integer solution. Since also the SD-MBPP model failed to find a feasible solution for some of the instances with a larger number of containers, in these cases we were not able to initialize the TS (in particular, this happened for the instances 4 and 6 denoted with (*) in Table 4). For this reason we decided to implement the LH described in Section 3.

Table 4 and 5 show the results produced in the two steps of the TS, LH-TS and those produced by the ACO algorithm. In both tables, the first group of columns reports the characteristics of the initial solutions obtained respectively by the SD-MBPP model (*SD-MBPP*) in Table 4 and by the LH (*LH*) in Table 5; then, the second group of columns shows the final average results respectively for the TS approach (*TS*) in Table 4 and the LH-TS one (*LH-TS*) in Table 5. Note that, as the tabu search is only able to improve the stability and the loading time of a solution, the number of non loaded containers (*NLC*) in the initial solutions found by the SD-MBPP model and LH is not modified in the final TS and LH-TS solutions. The last column of Table 5 shows the non loaded containers (NLC) in the final ACO solution together with the loading time and the CPU time.

Table 4 and 5 report the stability violations (σ_1 and σ_2) only for the initial solutions as no stability violation were found in the final solutions; this is true also for the ACO solution.

The *LT (i)* and *LT (f)* columns show the loading times respectively for the initial and final solutions, and the *LT %var* column reports the percentage variation of the loading time in the final solutions with respect to the initial ones. Finally, Table 4 and 5 include three CPU times expressed in seconds: the one needed by the SD-MBPP model (*CPU*), the time required by the tabu search (*CPU TS*) and the overall (initialization plus search) cumulative time (*CPU TOT*). Note that the averages in the last row of Table 4 are computed without instances 4 and 6. The ACO needed a shorter CPU time than the TS procedure, always being able to find a feasible solution. However, also the ACO approach was able to completely locate all the

containers only in one case out of 11; anyway, the number of non loaded containers is lower than those obtained by TS.

Table 4: Tabu Search solutions for the large size tests.

	SD-MBPP					TS (average over 5 runs)			
	NLC	**LT (i)**	σ_1	σ_2	**CPU**	**LT (f)**	**LT %var**	**CPU TS**	**CPU TOT**
1	5	868400	0	40	14400.0	868388	0.00	25.9	14425.9
2	0	852590	0	735	14400.0	853002	0.05	100.4	14500.4
3	100	850070	0	335	14400.0	850194	0.01	33.0	14500.4
4	84 (*)	896760 (*)	-	-	-	-	-	-	-
5	92	906310	5	300	11099.3	906288	0.00	22.3	11121.7
6	85 (*)	915329 (*)	-	-	-	-	-	-	-
7	112	918920	20	425	7325.2	918996	0.01	48.5	7373.7
8	47	901750	0	270	4181.7	901760	0.00	30.5	4212.2
9	194	733240	35	150	10187.8	732640	-0.08	37.4	10225.1
10	213	727830	20	85	14953.3	727510	-0.04	40.2	14993.5
11	91	842650	5	60	157.3	842616	0.00	25.3	182.6
Averages	95	844640			10122.73	844599		40.38	10163.12

Table 5: LH-TS and ACO solutions for the large size tests.

	LH					LH-TS				ACO		
ist	**NLC**	**LT (i)**	σ_1	σ_2	**CPU**	**LT (f)**	**LT %var**	**CPU TS**	**CPU TOT**	**NLC**	**LT**	**CPU**
1	240	897400	140	175	10	833640	-7.10	91.99	102.0	102.0	879162	154.8
2	188	896860	50	425	14	841420	-6.18	144.8	158.8	77.4	896676	123.4
3	424	855200	140	435	10	792606	-7.32	94.7	104.7	321.0	851946	100.9
4	106	972000	90	695	12	926130	-4.72	92.0	104.0	55.6	964970	73.1
5	44	991190	395	4080	15	969842	-2.15	80.5	95.5	0.0	986150	195.9
6	95	993200	25	4100	12	965824	-2.76	85.5	97.5	65.2	978940	130.1
7	100	996540	360	2975	14	946722	-5.00	160.8	174.8	19.6	999440	201.4
8	90	966160	230	515	10	935508	-3.17	94.7	104.7	1.4	979046	109.1
9	195	820640	80	1425	12	765732	-6.69	107.3	119.3	162.6	787590	166.7
10	219	815170	130	2315	11	758280	-6.98	107.5	118.5	152.6	799750	169.9
11	123	907340	625	660	10	882852	-2.70	95.6	105.6	40.0	918210	78.7
Avg	165.82	919245			11.82	874414.8		105.02	116.85	90.67	912898	139.36

Finally, comparing the results obtained by applying the proposed approaches to large size instances, in term of percentage number of non loaded containers (*NLC)* and the CPU time, we noted that none of the proposed approaches was able to load on board all containers: as the large ship test instances were randomly generated with occupancy level between about 87% and 97% to stress the solution capacity of the compared algorithms, we cannot claim that a solution where all the containers are loaded exists for this benchmark. What we can observe for the large size tests is that all the proposed methods produced feasible solutions in terms of stability conditions, and the ACO approach behaved better since it required lower CPU time and produced a lower average NLC.

5. Conclusions

In this paper we have presented and compared different solution methods for a very difficult problem that is the MBPP. The results here showed enable us to indicate that for very large size instances the ACO approach is recommended among the proposed ones. Moreover,

during our computational experiments we noted that the performance of TS based approaches are strongly depending on the goodness of the initial solution. For small and medium size instances we are able to apply the TS to a good initial solution, thus permitting us to obtain very good results from the TS. This is not the case for large size ones.
Starting from the results discussed in this paper, the authors intend to extend the obtained results to the multi-port MBPP. In this case we intend to study the problem of loading container by taken into account the whole trip of the ship, that is to say by considering the next ports that will be visited by the ship, where unloading and loading operations will be executed. Finally, in this future development the assumption concerning the usage of only one quay cranes will be removed.

References

1. Ambrosino D, Sciomachen A, Tanfani E.: Stowing a containership: The Master Bay Plan problem. Transportation Research 38, 81-99, 2004.
2. Ambrosino D, Sciomachen A, Tanfani E.: A decomposition heuristics for the container ship stowage problem, Journal of Heuristics 12, 211–233, 2006.
3. Ambrosino D, Anghinolfi D, Paolucci M, Sciomachen A. A new three-step heuristic for the master bay plan problem. Maritime Economics & Logistics, Special issue on "OR models in Maritime Transport and Freight Logistics 11(1), pp 98-120, 2009.
4. Anghinolfi D, Paolucci M.: A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. International Journal of Operations Research 5(1), 1-17, 2008.
5. Avriel M, Penn M, Shpirer N.: Container ship stowage problem: complexity and connection to the colouring of circle graphs. Discrete Applied Mathematics 103, 271-279, 2000.
6. Chen C.S., Lee S.M., Shen Q.S.: An analytical model for the container loading problem. European Journal Of Operation Research, 80, 1, 68-76, 1995.
7. Dorigo M, Blum C.: Ant colony optimization theory: A survey. Theoretical Computer Science 344, 243-278, 2005.
8. Dorigo M, Gambardella LM.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1, 53–66, 1997.
9. Dorigo M, Stützle T.: The ant colony optimization metaheuristics: algorithms, applications and advances. In Glover F, Kochenberger G (eds), Handbooks of metaheuristics, vol. 57, pp 252-285. Int. Series in Operations Research & Management Science. Kluver: Dordrech 2002.
10. Imai A, Nishimura E, Papadimitriu S, Sasaki K.: The containership loading problem. International Journal of Maritime Economics 4, 126-148, 2002.
11. Stahlbock R, Voss S.: Operations research at container terminal: a literature update. OR Spectrum 30, 1-52, 2008.
12. Steenken D, Voss S, Stahlbock R.: Container terminal operation and Operations Research - a classification and literature review. OR Spectrum 26, 3-49, 2004.
13. Stützle T, Hoos HH.: Max-min ant system. Future Generation Computer System 16, 889–914, 2000.