CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Optimizing Yard Assignment at an Automotive Transshipment Terminal

Jean-François Cordeau
Gilbert Laporte
Luigi Moccia
Gregorio Sorrentino

June 2010

CIRRELT-2010-28

Bureaux de Montréal :

Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :

Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Optimizing Yard Assignment at an Automotive Transshipment Terminal

Jean-François Cordeau[1,2], Gilbert Laporte[1,3], Luigi Moccia[4,*], Gregorio Sorrentino[5]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

[3] Canada Research Chair in Distribution Management, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

[4] Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Richerche, Via P. Bucci 41C, 87036 Rende (CS), Italy

[5] Dipartimento di Elettronica, Informatica e sistemistica, Università della Calabria, Via P. Bucci 41C, 87036 Rende (CS) Italy

Abstract. This paper studies a yard management problem in an automotive transshipment terminal. Groups of cars arrive to and depart from the terminal in a given planning period. These groups must be assigned to parking rows under some constraints resulting from managerial rules. The main objective is the minimization of the total handling time. Model extensions to handle application specific issues such as the rolling horizon and a manpower leveling objective are also discussed. The main features of the problem are modeled as an integer linear program. However, solving this formulation by a state-of-the-art solver is impractical. In view of this, we develop a metaheuristic algorithm based on the adaptive large neighborhood search framework. Computational results on real-life data show the efficacy of the proposed metaheuristic algorithm.

Keywords. Yard management, automotive transshipment terminal, adaptive large neighborhood search.

Acknowledgements. This work was partly supported by the MUR (Italy) under project AUTOMA, and by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 227837-09 and 39682-10. This support is gratefully acknowledged. The authors thank the BLG Italia managers Francesco De Bonis and Francesco Scalise for their kind availability in explaining the application, and Alessandra De Luca for her help in defining the instance set. Special thanks are due to Manlio Gaudioso who initiated the cooperation framework that made this research possible.

---

* Corresponding author: Luigi.Moccia@cirrelt.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2010

1 Introduction

The purpose of this paper is to model and solve the problem of assigning cars to parking *rows* in an automotive transshipment terminal. Maritime automotive transportation is developing along the lines of container transportation where the hub and spoke arrangement is widely adopted (Mattfeld, 2006). Deep-sea vessels operate between a limited number of transshipment terminals called hubs. Smaller feeder vessels link the hubs with the other ports which are the spokes of the system. This network topology results in the consolidation of capacity along the routes connecting the transshipment ports, and in the growth of their importance. Deep-sea car carriers have a capacity of up to 6000 vehicles whereas the capacity of ships deployed on short-sea segments can attain 1000 vehicles. Therefore, automotive transshipment terminals manage large flows of incoming and outgoing cars. Unlike containers, cars are considered to be fragile objects that require careful and consequently labour intensive handling. For example, cars cannot be stacked, which results in larger yards compared with container terminals. Cars must be parked in a yard made up of rows of varying lengths. Once assigned to their parking row, the cars remain in the same yard position for the duration of their stay in order to reduce the risk of damage. This "no-relocation" rule, combined with the low density of the yard, increases the importance of optimal yard assignment. Cars are transported from the quay to their parking slot by drivers who are grouped in teams and are transported by a mini-bus that brings them back to their starting point. In the following a set of cars that arrive and depart by the same vessel pair, and are of the same type (model and brand) will be called a *group*. To facilitate the yard management and the driver busing process, a group is allocated to a set of adjacent parking rows. The number of required rows depends on the car length and on the row length. Yard managers prefer not to share a row between different groups, which often results in partially empty rows.

This study was motivated by an application at the BLG Italia automotive transshipment terminal which operates in the port of Gioia Tauro located in southern Italy, on the West coast. Its barycentric position in the Mediterranean Sea makes this port very attractive as a hub terminal (see Monaco et al. (2009) for a discussion about the Gioia Tauro container terminal). The automotive terminal handles 75,000 cars per year. The yard is spread over an area of 11 ha, and its 374 parking rows have lengths varying from 50 to 70 meters. Figure 1 provides an aerial view of the terminal and highlights the two main yard areas. Mother vessels unload cars while berthing at the quay on the right of Figure 1, and feeder vessels load cars while berthing at the quay on the left of Figure 1.

Figure 1: Aerial view of the BLG Italia terminal in the port of Gioia Tauro

The remainder of this paper is organized as follows. We present in Section 2 an optimization model for the yard allocation process and we analyze the computational complexity of the problem. Two integer linear programming formulations and model extensions are discussed. The relationships between the yard assignment problem and other known problems are investigated in Section 3. We describe in Section 4 a metaheuristic algorithm for our problem, while Section 5 presents computational experiments followed by a conclusion.

2 Optimization model

We first introduce the notation used to derive integer linear programming formulations for our problem. We then discuss the computational complexity of the problem and some extensions.

2.1 Notation

The problem is defined on a time horizon discretized in $|T|$ time steps indexed by $t \in T = \{1, ..., |T|\}$. The set of groups to allocate during the time horizon is indicated by $K = \{1, ..., |K|\}$, and $R = \{1, ..., |R|\}$ is the set of parking rows. The data related to groups are:

- n^k, number of cars in group k;
- v_r^k, maximum number of cars of group k that can fit in row r;
- $a^k \in T$, arrival time of group k;

- b^k, departure time of group k;
- o^k, quay unloading position of group k;
- d^k, quay loading position of group k;
- c_a^k, largest admissible handling time when unloading group k;
- c_b^k, largest admissible handling time when loading group k.

The groups considered in set K are those arriving inside the time horizon, and the departure time of a group may exceed the time horizon.

Rows are numbered in their filling direction, i.e. if row r is filled before row s then $r < s$. The row ordering is such that if rows r and s are adjacent and $r < s$, then $s = r+1$. We will consider later in this section the case of an "ending-row" arising when a given row r does not have an adjacent row in the filling direction. In the following we assume that there always exists an adjacent row. For each group k we have to find a set of free adjacent rows of sufficient capacity. Since we consider parking rows of varying lengths, the number of required rows is variable as well. In other words, if r is the first row in the filling direction assigned to group k, then the last row will be $r + q_r^k - 1$, where q_r^k is the smallest positive integer value satisfying

$$\sum_{\alpha=0}^{q_r^k-1} v_{r+\alpha}^k \geq n^k.$$

The q_r^k value expresses the number of rows needed by the group k when the first row of the group is r, i.e. the group would occupy the row interval F_r^k defined as $F_r^k = \{r, r+1, ..., r+q_r^k-1\}$. Analogously, we denote by u_s^k the number of rows that group k will require if s is the last row of the group, i.e. u_s^k is the smallest positive integer value satisfying

$$\sum_{\alpha=0}^{u_s^k-1} v_{s-\alpha}^k \geq n^k.$$

Consequently, we have the row interval $B_s^k = \{s-u_s^k+1, s-u_s^k+2, ..., s\}$ which is equivalent to F_r^k whenever $r = s - u_s^k + 1$. Since the q_r^k and u_s^k values are related to the filling direction, we refer to them as "forward row request", and "backward row request", respectively. For notational compactness, we define the following sets:

- $T(k) = \{t \in T : a^k \leq t \leq b^k\}, \forall k \in K,$ the set $T(k)$ represents the duration of stay of group k in the planning horizon;
- $K(t) = \{k \in K : t \in T(k)\}, \forall t \in T,$ groups that are in the terminal at time step t;
- $K_a(t) = \{k \in K : t = a^k\}, \forall t \in T,$ groups that arrive at the terminal at time step t;
- $K_b(t) = \{k \in K : t = b^k\}, \forall t \in T,$ groups that leave the terminal at time step t.

The set of feasible first row assignments for a group k is denoted by $R(k) \subseteq R$. The set $R(k)$ handles some aspects of the planning problem in a rolling horizon framework. In fact, the assignments must comply with rows occupied by groups already in the yard at the first time step. Therefore, these pre-assigned groups are taken into account in the definition of the sets $R(k)$.

The "ending-row" case is now treated by considering as infeasible an assignment of a group k to a first row r such that $q_r^k > 1$ and the set $\{r, r+1, ..., r+q_r^k-2\}$ contains an ending row. Let $\bar{R} \subset R$ be the set of ending rows. We define as $R(k)$ the subset of R such that there does not exist an intermediate ending row for any assignment of k to $r \in R(k)$, i.e. $R(k) = \{r \in R : F_r^k \setminus \{r+q_r^k-1\} \cap \bar{R} = \emptyset\}$.

With this notation we can characterize the assignment of a group by its assignment to a first row. Our decision variables are:

- $y_r^k \in \{0,1\}, k \in K, r \in R(k)$, $y_r^k = 1$ if the first row of the group k is r, i.e. the group occupies the row set $\{r, r+1, ..., r+q_r^k-1\}$.

The assignment of row r as the first row for group k, i.e. $y_r^k = 1$, forbids some assignments of groups to rows. The affected groups are those that are present in the yard during the stay of group k, i.e. groups h such that $T(k) \cap T(h) \neq \emptyset$. Any such group h cannot be assigned to any row s that interferes with group k. A forbidden row s for h is such that $F_r^k \cap F_s^h \neq \emptyset$. We define the set Φ as the set of quadruples (k, r, h, s):

$$\Phi = \{(k,r,h,s) : k, h \in K, h > k, r \in R(k), s \in R(h), F_r^k \cap F_s^h \neq \emptyset, T(k) \cap T(h) \neq \emptyset\}.$$

A quadruple (k, r, h, s) belonging to Φ indicates that the variables y_r^k and y_s^h cannot be both equal to one.

We now introduce the data required for the objective function of our problem. Since we want to minimize the total handling time, we define as c_{vz} the handling time required to move a car between $v \in R \cup O$ and $z \in R \cup D$, where the set $O = \bigcup_{k \in K}\{o^k\}$ represents the unloading positions. Similarly we indicate by $D = \bigcup_{k \in K}\{d^k\}$ the set of loading positions.

Our decision variables induce cost coefficients defined as follows:

- $c^k_{o^k r}$, unloading handling time for the group k when the first assigned row is r:

  $c^k_{o^k r} = \sum_{\alpha=0}^{q^k-2} c_{o^k, r+\alpha} v^k_{r+\alpha} + c_{o^k, r+q^k-1}(n^k - \sum_{\alpha=0}^{q^k-2} v^k_{r+\alpha})$.

- $c^k_{rd^k}$, loading handling time for the group k when the first row is r:

  $c^k_{rd^k} = \sum_{\alpha=0}^{q^k-2} c_{r+\alpha, d^k} v^k_{r+\alpha} + c_{r+q^k-1, d^k}(n^k - \sum_{\alpha=0}^{q^k-2} v^k_{r+\alpha})$.

These cost coefficients are used in the objective function. Observe that the loading handling time is defined for all groups, hence also for those leaving the terminal after the end of the planning horizon. Thus we account for a *future* loading handling time in the *current* planning horizon. Moreover, these cost coefficients are used to define the set of feasible assignments $R(k)$. A row r does not belong to $R(k)$ whenever $c^k_{o^k r} > c^k_a$ or $c^k_{rd^k} > c^k_b$. We observe that this models loading and unloading priorities. The c^k_a (respectively c^k_b) coefficient of a group k can be set to smaller values to ensure that the group k is assigned to rows closer to the unloading (respectively loading) quay position. This results in user-controlled parameters to specify group priorities, since closer rows mean shorter handling times.

2.2 Integer linear programming formulations

We can now formulate our problem, hence called the adjacent row dynamic assignment problem (ARDAP), by means of the following model $\mathscr{F}_1$:

$$\text{minimize} \sum_{k \in K} \sum_{r \in R(k)} (c^k_{o^k r} + c^k_{rd^k}) y^k_r \tag{1}$$

subject to

$$\sum_{r \in R(k)} y^k_r = 1 \qquad \forall k \in K, \tag{2}$$

$$y^k_r + y^h_s \leq 1 \qquad \forall (k, r, h, s) \in \Phi, \tag{3}$$

$$y^k_r \in \{0, 1\} \qquad \forall k \in K, \forall r \in R(k). \tag{4}$$

The objective function (1) minimizes the sum of the handling times. Constraints (2) state that each group k must be allocated to one and only one admissible first row r, since r must belong to $R(k)$. The feasibility of the assignment is guaranteed by constraints (3) which forbid pairs of incompatible assignments as defined by the set Φ.

The model uses $|K| \times |R|$ binary variables and the number of constraints is $O(|K| + |K|^2 \times |R|^2)$. We can obtain a more compact model $\mathscr{F}_2$ by replacing constraints (3) with

$$\sum_{k \in K(t)} \sum_{s \in B_r^k} y_s^k \leq 1 \qquad \forall r \in R, \forall t \in T. \tag{5}$$

Indeed, for a given $k \in K(t)$ the variables $y^k_{r-u^k_r+1}, \ldots, y^k_r$ are such that if one of them is equal to one, then the row r is used by group k as first row (the case $y^k_r = 1$), or as last row (the case $y^k_{r-u^k_r+1} = 1$), or as intermediate row in the other cases. Thus, constraints (5) state that if row r is used by a group at time step t, but not necessarily as a first row, then its use is forbidden for all other groups staying in the yard at that time step.

The new model $\mathscr{F}_2$ still has $|K| \times |R|$ binary variables, but the number of its constraints is now $O(|K| + |T| \times |R|)$. We found that model $\mathscr{F}_1$ can only solve small instances, whereas we are able to solve model $\mathscr{F}_2$ for larger instances. We will present this comparison in Section 5.

2.3 Computational complexity

In the following we prove that ARDAP is strongly $\mathscr{NP}$-hard.

Theorem 1 *ARDAP is strongly $\mathscr{NP}$-hard.*

Proof — We prove this result by showing that the generalized assignment problem (GAP), which is is strongly $\mathscr{NP}$-hard, is a particular case of the ARDAP. In the GAP the aim is to determine a minimum cost assignment of a set of weighted items to a set of knapsacks (Martello and Toth, 1992). Let $N = \{1, ..., n\}$ be the set of items, and $M = \{1, ..., m\}$ the set of knapsacks. We indicate by c_{ij} the assignment cost of item i to knapsack j, by w_{ij} the weight of item i when assigned to knapsack j, and by W_j the capacity of knapsack j. An equivalent ARDAP instance can be defined as follows:

- an item i corresponds to a group k and vice versa, i.e $K = N$, and in the following we equivalently refer to items or groups;

- the ARDAP time horizon consists of only one time step, $|T| = 1$, and all groups defined above arrive and leave the terminal at this time step, i.e. $K(1) = K$;
- the number of rows is equal to the sum of the knapsack capacities, $|R| = \sum_{j\in M} W_j$;
- we partition the set R into m subsets S_j, $j \in M$: $S_j = \{r_j, ..., s_j\}$, where $r_j = \sum_{l=1}^{j-1} W_l + 1$ and $s_j = r_j + W_j - 1$, i.e. $|S_j| = W_j$; in the following we denote these sets as artificial knapsacks;
- the group forward row request q_r^k is constant for the row belonging to a given subset S_j, and it is equal to the corresponding weight of the item: $q_r^k = w_{kj}, \forall r \in S_j, j \in M$; similarly, the group backward row request u_s^k is equal to the weight of the item in each subset S_j;
- the group to row assignment cost $c^k_{o^k r} + c^k_{rd^k}$ is constant for the row belonging to a given subset S_j, and it is equal to the corresponding cost of the item, i.e $c_{kj}, \forall r \in S_j, j \in M$;
- the set $R(k)$ is constructed so as to avoid assignments of group k to rows that would exceed the capacity of the artificial knapsack: row $r \notin R(k)$ if there are two artificial knapsacks j and l such that $F_r^k \cap S_j \neq \emptyset$ and $F_r^k \cap S_l \neq \emptyset$.

The procedure outlined above constructs an ARDAP instance equivalent to a GAP. An optimal solution for this ARDAP instance could be polynomially transformed into an optimal solution for the GAP. Therefore, if there existed a pseudo-polynomial algorithm $\mathscr{A}$ for the ARDAP, then $\mathscr{A}$ would solve the GAP as well. Since the GAP is known to be strongly $\mathscr{N}\mathscr{P}$-hard, the result follows. $\square$

2.4 Extensions of the model

In real-life, yard assignment decisions are made on a daily basis by the yard planner who knows with a high degree of reliability the list of calling vessels and the groups of cars that will arrive and depart within a planning horizon of one week. Data regarding the following weeks are considered to be insufficiently reliable for yard planning. Every day the planner assigns the groups expected to arrive within the planning horizon, but assignment for an incoming group can change between two subsequent plans. The final assignment is determined upon the arrival of the group. In this sense, the yard management operates according to a *rolling horizon* framework. This dynamic setting, and the favorite policy of assigning a group to adjacent rows can cause infeasibilities because of yard fragmentation.

This occurs whenever the number of free rows is at least equal to the number of requested rows, but is insufficient to park the cars according to the favorite "adjacent rows" policy. Whenever this situation occurs the yard planner must determine a configuration amenable to the favorite policy. He can decide to break an incoming group into smaller ones, or to relocate some groups of cars. This last option is the least preferred and is avoided as much as possible.

We have devised a modification of the objective function in order to consider this issue. The idea consists in favoring yard plans that have a large set of free adjacent rows at the end of the planning horizon. Thus, the fragmentation risk is minimized when the new plan is drawn on the following day. Let L_t be the largest total length of free adjacent rows at time step t in a given yard plan. Then, the modified objective function is

$$\text{minimize} \sum_{k \in K} \sum_{r \in R(k)} (c^k_{o^k r} + c^k_{rd^k}) y^k_r - \gamma_1 L_{|T|}, \tag{6}$$

where $\gamma_1 > 0$. In Section 5 we will highlight the tradeoff between minimizing handling times and minimizing fragmentation.

The yard planner faces another set of issues related to manpower planning. Whenever the handling activities are low, he can choose assignments of incoming groups to less favorable positions, i.e. more distant ones. This strategy could result in an advantage because positions that are closer to the quay, and thus more favorable, are left free for busier periods. It is then of paramount importance to profile the level of handling activity in the terminal as a result of the yard allocation process. The concept of resource profile of planning activities upon shared terminal resources was introduced by Won and Kim (2009), and was also used by Giallombardo et al. (2010) for quay cranes in berth allocation plans. At time t the total handling induced by yard allocation is equal to $\sum_{k \in K_a(t)} \sum_{r \in R(k)} c^k_{o^k r} y^k_r + \sum_{k \in K_b(t)} \sum_{r \in R(k)} c^k_{rd^k} y^k_r$. We have added the following additional term to the objective function in order to obtain "close to desired" handling profiles:

$$\gamma_2 \sum_{t \in T} \left[\sum_{k \in K_a(t)} \sum_{r \in R(k)} c^k_{o^k r} y^k_r + \sum_{k \in K_b(t)} \sum_{r \in R(k)} c^k_{rd^k} y^k_r - H_t \right]^+ . \tag{7}$$

Here we indicate by H_t the largest desired handling value at time t. Thus (7) is the sum of the positive deviation from the desired handling profile. The positive weighting factor γ_2 is used to control the relative importance of this term of the objective.

The model can be solved iteratively by using arbitrarily large H_t values at the first iteration, which in fact disables the term (7). Then, if the planner prefers to smooth the resulting handling peaks of this first solution, the model is solved by imposing the desired H_t values. The process is iterated until a feasible and satisfactory solution has been found.

The model extensions just introduced suggest an iterative use of the model under different assumptions and input data such as group priorities, forecasts, desired fragmentation, desired handling profile, etc. The modified objective function (6), plus the term (7), could be incorporated within the integer linear programming formulation by adding proper variables and constraints. However, solving this problem exactly is impractical even for the basic model because of its computational complexity. These are additional motivations for the metaheuristic algorithm presented in Section 4.

3 Relations with other optimization problems

The scientific literature related to the management of container terminal yards is rich and expanding. For reviews see Vis and Koster (2003), Steenken et al. (2004), and Stahlbock and Voß (2008). The wide range of contributions is justified by the variety of technological configurations, of decision levels (strategic, tactical, operational, real-time), and of types of container flows (import, export, transshipment). Automotive terminals can be seen as another type of technological setting. The distinguishing features of yard management in automotive terminals with respect to container terminals have been discussed in Mattfeld and Kopfer (2003) and Mattfeld (2006). These features derive from the no-relocation policy, and the low density yard in this type of terminals. Therefore, container terminal based approaches cannot be straightforwardly applied to this context.

The work of Mattfeld and Orth (2007) is the closest to our study. These authors present a task scheduling and allocation problem in a large automotive terminal under different assumptions than ours. They differentiate between inbound storage tasks and outbound retrieval tasks. Both types of tasks must be executed within given time windows. This flexibility is exploited to handle the objective of leveling manpower utilization. This feature does not arise in our application because the terminal in our case deals mainly with vessel-to-vessel flows, and the arrival and departure times of groups are input data. Furthermore, in Mattfeld and Orth (2007) the space allocation is modeled at a more aggregate level than in our model (with knapsack type capacity constraints). This is justified by the larger and more complex layout of the terminal of Bremerhaven which serves as a basis for their study. The smaller size of the Gioia Tauro terminal enables us to optimize the assignment of cars to parking rows under the favorite policy of adjacent rows for groups.

The ARDAP can be also viewed as a variant of the two-dimensional rectangle packing problem. An assignment of a group to a set of yard rows is represented by a rectangle with the duration of stay as height and the number of occupied rows as width. The two dimensions of a bin are rows and time, and there are as many bins as the number of ending rows. Thus, solving the ARDAP is equivalent to packing a set of rectangles in several bins so that placement cost is minimized. Figure 2 illustrates the solution of an ARDAP instance with 20 groups to be allocated in a yard with 374 rows (horizontal axis) in a time horizon of 31 time steps (vertical axis), and with only one ending row (resulting in one bin). In this instance group 8 arrives at time 0, departs at time 9, and occupies rows 0 to 57. However,

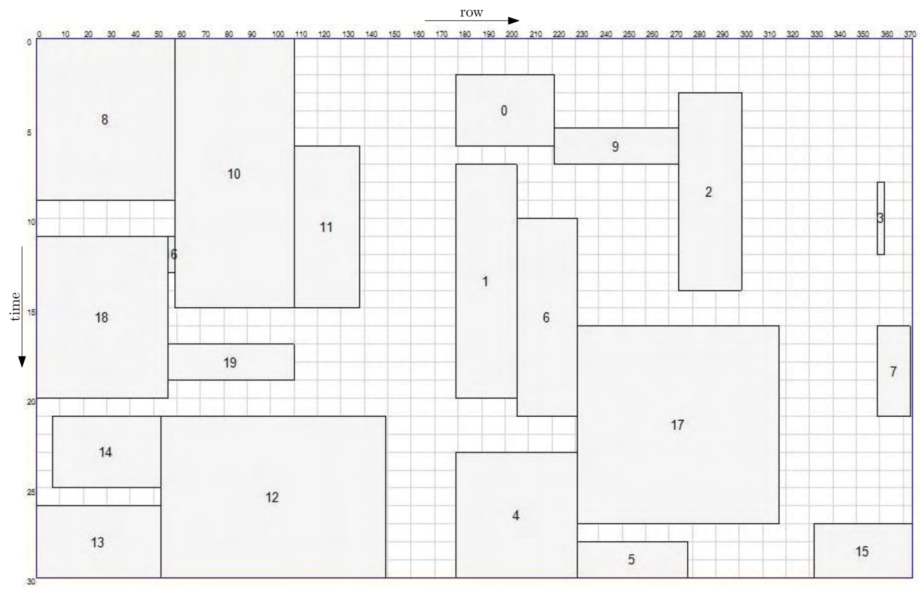

Figure 2: Optimal solution of an ARDAP instance in the row-time plane

the ARDAP exhibits many differences with respect to classical rectangle packing problems. The most popular of these problems, see e.g. Lodi et al. (2002), are the bin packing problem (BPP), and the strip packing problem (SPP), where the objective function to be minimized is the number of bins (for the BPP), or the height of the strip containing all rectangles (for the SPP). The ARDAP objective function is different because for each rectangle placement there is a position specific cost whose sum must be minimized. We also point out that the placement cost along the row axis is non-convex in our application and is specific for each group (i.e. for each rectangle). In the following we provide an illustrative example which needs some additional information about the application context.

The instance depicted in Figure 2 is derived from historical data of the operational

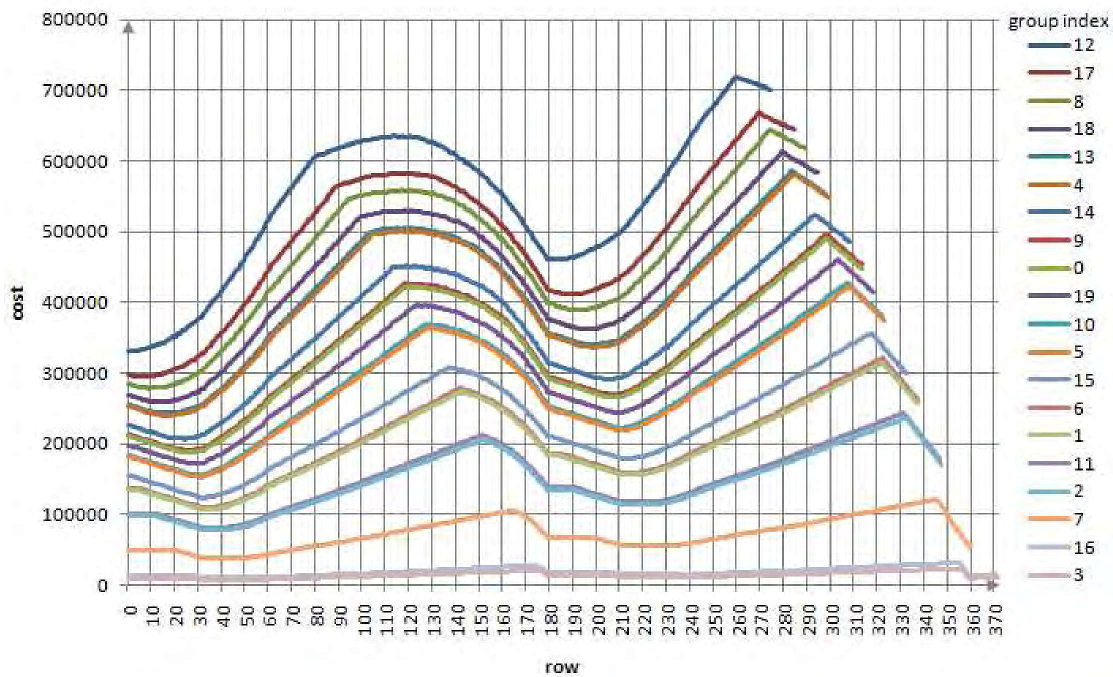

Figure 3: Cost function for each group of the instance depicted in Figure 2

database of the Gioia Tauro terminal. The assignment cost function is obtained considering the road network of the yard, and Figure 3 depicts these cost functions for all groups of this instance. Figure 4 illustrates the yard layout with the parking rows. Filled parking rows are represented as gray shaded. A similar shade intensity between adjacent rows indicates that these rows are assigned to a group of cars. Cars are usually unloaded from vessels berthing at the North quay (where the vessel on the right of Figure 4 is berthed) and loaded in vessels at the East quay (on the left of Figure 4). This results in unloading and loading costs for each assignment of groups to sets of adjacent rows. In Figure 4 we denote as area A_i and area B_i the set of rows relative to the same quay segment, and A_i is closer to the quay than B_i is. The set of rows are numbered in increasing order from left to right of Figure 4, with the exception of area B_0 which is a special area. This same ordering is applied to the numbering of individual rows and the rows of type A have a lower index than those of type B. In order to relate the cost functions of Figure 3 with the layout of Figure 4 we mention that the row numbered as 178 is the last row of area A_6 on the right of Figure 4. In fact, this row could be considered as an ending row not adjacent to the row 179 which is in the area B_1 on the left of Figure 4. Considering ending rows would cause discontinuities in the cost functions. We preferred to omit ending rows in the example for simplicity. This is not arbitrary because the yard planner often does not enforce the ending row concept and some groups are allocated following this numbering order.

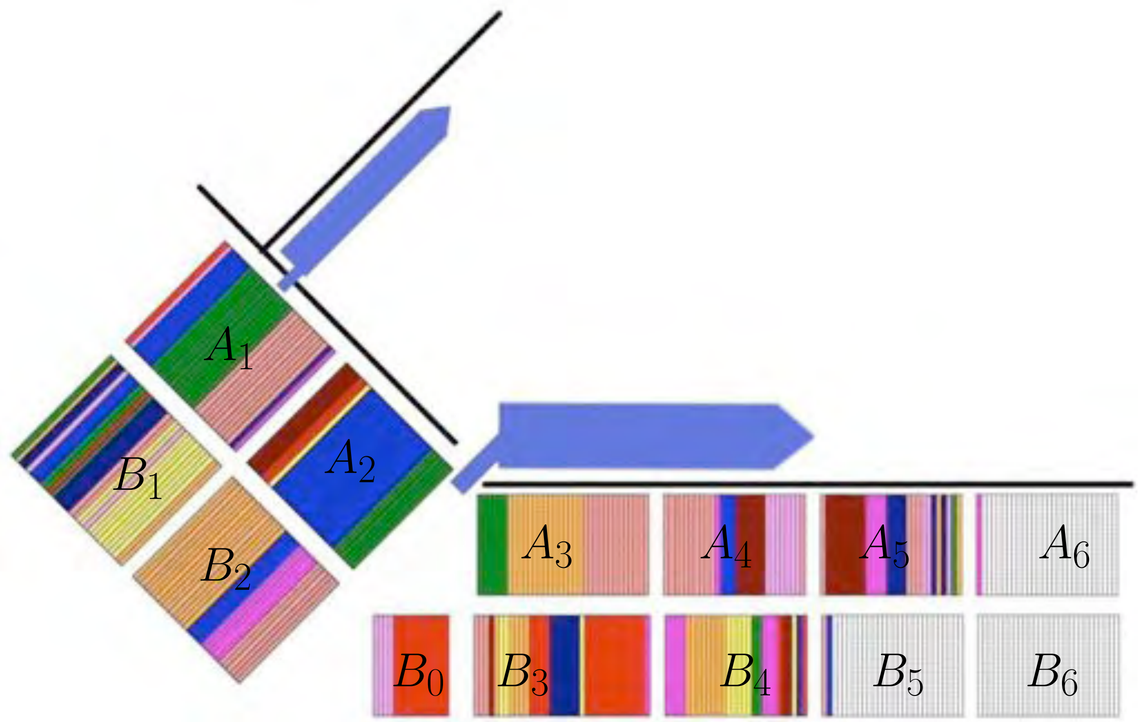

Figure 4: Example of yard allocation at the BLG Italia terminal

We can now discuss the impact of these rules on the solution method. The usual search strategy in rectangle packing heuristics explores the space of contiguous rectangles because this allows area minimization, but this could result in non-optimal solutions for the ARDAP. The solution of Figure 2 is optimal even though some rectangles are not contiguous. The optimal solution tends to assign some rectangles close to positions that would minimize their cost function, whereas other rectangles are "sacrificed" with different placements that tend to be as close as possible to other locally minimum positions. This issue is not particular to this problem. For example, it also occurs in berth allocation problems where yard costs are considered in the objective function. For the berth allocation problem see, e.g. Park and Kim (2003), Cordeau et al. (2005), Meisel and Bierwirth (2009), and for a recent survey Bierwirth and Meisel (2010). This is the reason that motivated us to develop a metaheuristic algorithm that looks for the explicit minimization of the cost function instead of the used area.

The ARDAP is more similar to the rectangle packing problem with general spatial costs introduced by Imahori et al. (2003) and Imahori et al. (2005). However, the ARDAP exhibits distinctive features: the width of a rectangle is dependent on the assigned position (the q_r^k values in our notation), and the placement cost also depends on the assigned position. The general packing problem of Imahori et al. (2003) and Imahori et al. (2005) could handle the ARDAP artificially by expanding the number of modes of a rectangle. In the framework of these authors, a rectangle can have different modes, i.e. dimensions, and mode specific costs. The ARDAP could be modeled by introducing a mode (r,k) for each $r \in R(k)$, and assigning a sufficiently high value to the cost of assigning a mode (r,k) to a row l with $l \neq r$. However, it is clear that these ARDAP specific features (position dependent rectangle dimensions and position specific costs) render the adaption of an existing algorithm for the rectangle packing impractical. Another ARDAP feature which makes the problem more constrained is that a rectangle can only move horizontally. The vertical placement cannot change since the arrival and departure times are fixed. In view of this, we are interested in exploiting the features of our problem. In particular, we can use the information of given duration of stay of groups. We have thus defined search schemes using this information.

The literature concerning storage assignment for automatic warehouse systems is also relevant to our problem. For example, an implicit assumption in yard management is the use of a *shared storage policy* (as opposed to a dedicated storage policy) whose merits have been discussed in the seminal paper of Goetschalckx and Ratliff (1990).

In the previous paragraphs we have examined the relationships between the ARDAP and the GAP. Moccia et al. (2009) have introduced an extension of the GAP, called the dynamic GAP (DGAP). Like the ARDAP, the DGAP considers a discretized time horizon and associates a starting time and a finishing time with each task. The ARDAP is more constrained than the DGAP which allows relocation during the duration of stay of the tasks (groups), whereas the ARDAP does not. However, the main difference between the two problems consists in the degree of detail on the spatial allocation: whereas the DGAP has knapsack-like capacity constraints, the ARDAP defines adjacent row assignment for each group.

4 A metaheuristic algorithm for the ARDAP

In the following we introduce a metaheuristic algorithm based on the adaptive large neighborhood search (ALNS) framework. In the ALNS a number of simple heuristics compete to modify the current solution (Pisinger and Ropke, 2007). A master level layer adaptively selects heuristics to intensify and diversify the search. At each iteration a heuristic is chosen to destroy the current solution, and another is chosen to repair it. The new solution is accepted if it satisfies the criteria of the local search algorithm chosen at the master level. The ALNS framework can be applied to a wide class of optimization problems. The adaptive layer chooses the heuristics according to the *scores* obtained at previous iterations. We denote the past score of the heuristic H_i by π_i, and the probability of selecting the heuristic H_j is

$$v \frac{\pi_j}{\sum_{i=1}^{z} \pi_i}, \tag{8}$$

where z is the number of heuristics. Details about the scores will be given in Section 4.5. In order to solve a given optimization problem using the ALNS framework one needs to design some destroy and repair heuristics, as well as a local search framework at the master level. The destroy heuristics remove groups from the yard, and the repair heuristics try to insert them in new positions. Sections 4.1 and 4.2 describe these two sets of heuristics. In our ALNS implementation we use a two-phase mechanism. A first phase looks for feasibility only, whereas the second phase tries to obtain good quality solutions. The reason for this is that some of our destroy and repair heuristics are either useful for feasibility or for optimality. We have designed a first phase in which the criterion for choosing the destroy and repair combination of heuristics is fixed. The algorithm starts by assigning each group to a dummy position with a high assignment cost. This is the starting infeasible solution. The first phase ends when a feasible solution is obtained, or the maximum number of iterations has been reached, in which case no feasible solution has been identified and an error message is returned. After this first phase, the selection of the destroy and repair heuristics is guided by the adaptive heuristic selection mechanism described in Section 4.5. Each improving solution is refined by applying a post-optimization procedure to be presented in Section 4.3. The master level local search is described in Section 4.4. Algorithm 1 outlines the ALNS algorithm.

4.1 Destroy heuristics

A destroy heuristic takes as input a given solution x and determines the ω groups to remove from the yard, where ω is an input parameter. When selecting the group to be removed it is important to consider the *time relatedness* of groups. Two groups are considered time related if they are both in the yard at the same time step, i.e. if $T(k) \cap T(h) \neq \emptyset$. Figure 5 depicts an example of time related groups. The main idea is that by re-assigning a set of groups such that each group is time related to at least another group in the set increases the

Algorithm 1 ALNS algorithm

Construct a starting solution x
$x^* = x$
repeat
  Choose a destroy heuristic H^- and a repair heuristic H^+ according to a given rule (first phase), or according to the probability based on the previously obtained score π (second phase)
  Generate a new solution x' from x using the heuristics H^- and H^+
  if x' can be accepted **then**
    $x = x'$
    Update score
    if $f(x) < f(x^*)$ **then**
      Apply a post-optimization procedure
      $x^* = x$
    end if
  end if
until stopping criterion is met
if x^* is feasible **then**
  return x^*
else
  return an error message
end if

likelihood of improving the objective function value.

In order to select the groups to be removed, the list of groups is sorted according to some criterion. The groups are then chosen by scanning the list and selecting a group with probability p, where p is an input parameter in the interval $(0,1]$. This is accomplished by randomly drawing a random number ρ in $(0,1]$ and comparing it with p. Whenever $\rho \leq p$ the current group is chosen from the list; otherwise the next group in the list is considered. The parameter p plays a randomization role. Whenever $p = 1$ the groups are selected exactly as in the list order. A smaller value of p increases the probability of choosing groups further down the list. The removed groups are added to a *destroy set* to be used by the repair heuristic that will try to reallocate them. Removing a group from the yard corresponds to releasing the yard rows previously assigned to that group. Algorithm 2 outlines a generic destroy heuristic. We use four destroy heuristics that differ mainly in the sorting criterion of the list of groups. The first is used only at the beginning of the process, to find a feasible initial solution.

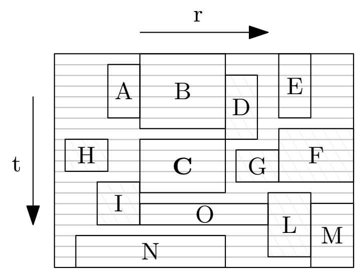

Figure 5: Example of time related groups in a space-time representation. D, F, G, H, I and L are the groups related to group C

Algorithm 2 Generic scheme of a destroy heuristic

Sort groups according to some criterion
repeat
  Select a group k scanning the sorted list according to the randomization parameter p
  Add k to the destroy set
  Remove k from the list, from the current solution x and free yard rows previously assigned to k
  Perform some heuristic specific operations
until ω groups are removed

4.1.1 Largest-out heuristic

The largest-out heuristic lists the groups in non-decreasing order of number of cars, which is a proxy for the number of rows that will be used by a group. The list is then scanned and a first group is selected with probability p. We denote the first chosen group as the *seed* group. As soon as a seed group is selected, groups that follow in the list are chosen only if they are time related to the seed group. If the end of the list is reached before selecting ω groups, then the procedure iterates by choosing another seed group among the remaining ones in the list.

4.1.2 Time-step destroy heuristic

The purpose of this heuristic is to remove a set of groups that are contemporary in the yard. The algorithm randomly chooses a time $\bar{t}$, and then creates a list of groups such that each group in the list is in the yard at time $\bar{t}$. Whenever the number of groups in this list is less than ω, all of them are added to the destroy set and the procedure iterates by choosing a different time step. Otherwise, exactly ω groups are selected from the list by using the randomization parameter p.

4.1.3 Worst-out heuristic

The worst-out heuristic is based on the *quality* of group-row assignments. For each group k we define the quality index $\triangle_{f_k}$ as the difference between the current assignment cost for the group k and the ideal cost f_k^*. The value f_k^* is the best assignment cost that the group k could have if the yard was entirely available, i.e. $f_k^* = \min_{r \in R(k)}\{c^k_{o^k r} + c^k_{rd^k}\}$. The heuristic first chooses a group k by scanning the list of groups ordered by non-increasing values of $\triangle_{f_k}$. It then chooses groups that are time related to k by scanning the list starting from a random position. The process is iterated until ω groups are selected.

4.1.4 Random removal heuristic

This is the simplest heuristic. It randomly selects ω groups to remove from the yard, the only condition being that a group must be time related to at least another group already in the destroy set. The aim of this heuristic is to diversify the search.

4.2 Repair heuristics

We now describe two greedy repair heuristics. As mentioned, the first heuristic is aimed at recovering feasibility. This heuristic is used in the first phase of the algorithm. The second heuristic tries to balance feasibility and cost minimization and is always applied in the second phase after a feasible solution has been found.

4.2.1 Largest-first heuristic

The largest-first heuristic exploits as greedy principle the *dimension* of a group, defined as the product of the total length of the cars in a group and their duration of stay expressed in number of time steps. This value is a proxy for the area of the rectangle in the row-time plane. Observe that in this plane the spatial dimension is variable, whereas this dimension index has the advantage of being fixed. The groups are assigned to the yard in non-increasing order of this index and according to the randomization parameter p. Every selected group is assigned to the first feasible position in the yard filling direction. The aim of this heuristic is to minimize the used area in the row-time plane, which helps reaching feasibility.

4.2.2 Worst-first heuristic

The worst-first heuristic favors the worst allocated groups in order to give them the opportunity to be assigned first. This algorithm orders the groups by non-increasing values of $\triangle_{f_k}$. Groups are chosen by scanning the list with the randomization parameter p. Once selected, a group is assigned to the yard at the minimum cost feasible position with probability p, otherwise it is assigned to the first feasible position in the yard filling direction. Thus, this heuristic looks at both cost and area minimization.

4.3 Post-optimization procedure

Every time an improving solution is found we apply a post-optimization process that removes and then reassigns each group to a more favorable position, if any. Groups are selected according to non-increasing values of $\triangle_{f_k}$. The process is iterated until, after a complete scan of the sorted list of groups, no more improvement has been registered.

4.4 Master level local search

At the master level we choose to use simulated annealing as local search framework. We accept a newly identified candidate solution x', given a current solution x, with probability

$$e^{-(f(x')-f(x))/\tau}, \tag{9}$$

where τ is the temperature which starts from τ_{start} and decreases at each iteration i according to the expression $\tau_i = c\tau_{i-1}$, and $0 < c < 1$ is the cooling rate. We only accept new solutions that have not previously been accepted.

4.5 Adaptive heuristic selection mechanism

As mentioned, we have designed a first phase in which the criterion for choosing the destroy and repair combination of heuristics is fixed and it results in the application of the largest-out (Section 4.1.1) and the largest-first (Section 4.2.1) pair of heuristics. This is because the largest-out and the largest-first heuristics is a destroy and repair combination that excels at feasibility, whereas the other destroy heuristics and the worst-first repair heuristic (Section 4.2.2) are particularly useful for generating good quality solutions. The adaptive selection mechanism of heuristics is used in the second phase of the algorithm. It is based on the scores π_j assigned to each destroy heuristic. The repair heuristic is always the worst-first heuristic. To select the destroy heuristics, we collect the scores, as suggested by Pisinger and Ropke (2007), over a segment of 100 iterations. The score π_{ij} of a heuristic i in a segment j is obtained from the score in the previous segment incremented, at each iteration, with the following values depending on the new obtained solution x':

- σ_1, if x' is a new best solution;
- σ_2, if x' is better than the current solution;
- σ_3, if x' is worse than the current solution, but it is accepted.

Since we accept only solutions not accepted before, a long term memory is needed in order to keep track of all solutions already accepted.

5 Computational experiments

We now present computational experiments. We first describe the generation of test instances, we then provide implementations details, and finally we discuss results obtained with the metaheuristic and with a commercial integer linear programming solver applied to the proposed formulations.

5.1 Generation of test instances

We have generated a set of 63 instances for the ARDAP problem using real-life data of the Gioia Tauro terminal. We have considered a time step of one day and a time horizon of 31 days. This results in a planning horizon of one month, i.e. four times larger than the usual horizon of one week. The reasons for this choice are the following:

- A longer planning horizon gives more challenging instances.
- Terminal expansion and volume increase could occur in the future, and would result in more difficult yard assignment problems.
- The current practice of planning with a time step of one day could also change. Container terminals normally plan with a time step equal to the length of a work shift, which results in four time steps per day, usually. If the Gioia Tauro terminal were to adopt this practice, the number of time steps per week would be equal to 28, close to the number considered in the generated instance set.

- Another change that could require solving larger instances in the terms of the time horizon would be the inclusion in the planning of forecasts for weeks following the current one.

Furthermore, we observe that one-month instances are useful to assess the effect of the rolling horizon. By solving the full instance of 31 days we obtain a lower bound on what can be achieved by solving smaller problems with a one-week time horizon for each day of the month.

At the time of our study, average speeds between yard positions where not available in the terminal operational database. We have used physical distances as proxies for handling times. The yard layout is the one described in the Introduction. Each instance was generated by fixing the number of groups, and randomly generating the values for the numbers of cars and the duration of stay for each group according to discrete uniform distributions within historical ranges. Arrival times were considered to be uniformly distributed in the time period of 31 days, and every group must leave the terminal during this period. Tables 1 and 2 report average, minimum, and maximum values for a set of characteristics of the generated instances. We indicate by yard saturation degree at a time step the ratio between the total length of rows required to allocate cars at that time step, and the total length of the parking rows in the yard. The total length of required rows at a time t is computed by assigning the groups in the set $K(t)$ to consecutive parking rows. The sum of the length of the used row defines the total length of required rows at the time t. This is clearly an optimistic saturation index because it does not account for interferences in the spatial allocation due to the duration of stay of the groups. These 63 instances are the feasible ones of a larger set. Feasibility was established by running one of the proposed formulations on the instance set.

	Instance	Yard saturation degree per time step			Number of cars per time step			Duration of stay per group (days)			Number of cars per group		
$\lvert K \rvert$	index	avg	min	max	avg	min	max	avg	min	max	avg	min	max
20	1	0.63	0.10	0.85	2390	371	3238	8	2	14	410	27	958
	2	0.61	0.15	0.85	2327	564	3261	7	3	16	476	83	993
	3	0.62	0.20	0.85	2372	802	3254	6	2	15	503	60	930
	4	0.65	0.21	0.86	2490	836	3288	7	1	16	488	13	964
	5	0.67	0.20	0.86	2557	790	3297	8	2	15	449	30	973
	6	0.64	0.21	0.86	2451	820	3294	6	2	15	501	12	948
	7	0.57	0.23	0.74	2204	881	2839	7	1	16	439	13	916
	8	0.54	0.22	0.79	2053	853	3021	7	1	15	415	23	941
	9	0.59	0.35	0.86	2245	1338	3292	7	1	16	501	46	949
	10	0.58	0.21	0.78	2232	808	2967	7	2	16	440	26	936
	11	0.57	0.21	0.77	2185	838	2938	7	1	16	468	63	966
	12	0.59	0.32	0.80	2261	1224	3046	6	2	15	530	48	999
40	13	0.68	0.03	0.87	2584	122	3257	7	1	16	243	15	470
	14	0.64	0.02	0.85	2423	87	3212	7	1	16	239	20	487
	15	0.64	0.13	0.86	2423	491	3279	7	1	15	275	23	494
	16	0.64	0.13	0.86	2402	491	3276	7	1	16	242	10	497
	17	0.67	0.16	0.89	2530	613	3370	7	1	16	245	28	499
	18	0.63	0.02	0.92	2371	81	3490	7	1	16	238	10	481
	19	0.57	0.20	0.76	2142	767	2876	7	1	16	229	11	482
	20	0.59	0.08	0.73	2242	305	2783	7	1	16	222	10	465
	21	0.56	0.12	0.81	2110	444	3054	6	1	15	239	13	484
	22	0.62	0.14	0.78	2339	538	2961	7	2	15	243	12	483
	23	0.61	0.19	0.80	2325	732	3048	7	2	16	233	10	497
	24	0.56	0.06	0.81	2121	223	3078	7	1	15	229	10	481
50	25	0.56	0.12	0.77	2095	462	2893	7	1	16	160	20	290
	26	0.55	0.07	0.81	2092	255	3022	7	1	15	165	18	295
	27	0.58	0.01	0.80	2178	31	2987	7	1	16	162	11	298
	28	0.57	0.11	0.77	2151	423	2882	7	1	16	155	13	294
	29	0.56	0.06	0.82	2122	243	3065	8	1	16	145	12	293
	30	0.56	0.01	0.77	2101	49	2876	7	1	16	154	11	290

Table 1: Characteristics of the generated instances, part I.

	Instance	Yard saturation degree per time step			Number of cars per time step			Duration of stay per group (days)			Number of cars per group		
$\lvert K \rvert$	index	avg	min	max	avg	min	max	avg	min	max	avg	min	max
20	31	0.73	0.38	0.94	2785	1450	3615	7	2	15	572	14	997
	32	0.73	0.39	0.94	2771	1514	3604	9	3	16	466	46	956
	33	0.77	0.35	0.93	2947	1319	3572	7	2	16	554	56	965
	34	0.72	0.24	0.97	2765	930	3715	7	2	16	494	42	986
	35	0.72	0.22	0.93	2757	844	3543	7	2	15	527	68	972
	36	0.77	0.29	0.88	2935	1124	3381	8	2	14	472	67	985
	37	0.68	0.27	0.91	2625	1026	3475	6	2	14	529	17	935
	38	0.65	0.21	0.86	2490	836	3288	7	1	16	488	13	964
	39	0.67	0.30	0.90	2553	1169	3423	8	1	16	449	49	992
	40	0.69	0.26	0.91	2638	996	3474	7	1	15	545	26	970
	41	0.69	0.33	0.90	2652	1258	3427	7	2	16	533	13	916
	42	0.66	0.32	0.89	2532	1226	3419	7	3	16	465	22	893
40	43	0.73	0.10	0.97	2789	389	3690	7	1	16	281	41	496
	44	0.72	0.21	0.95	2742	828	3595	7	2	16	255	18	489
	45	0.73	0.24	0.91	2764	939	3435	8	1	16	263	12	495
	46	0.72	0.12	0.88	2740	454	3342	7	1	16	263	22	493
	47	0.72	0.18	0.98	2746	686	3714	7	2	16	269	18	489
	48	0.73	0.10	0.93	2754	383	3519	8	3	16	249	21	476
	49	0.68	0.03	0.87	2584	122	3257	7	1	16	243	15	470
	50	0.67	0.18	0.92	2534	705	3487	8	1	16	237	16	471
	51	0.73	0.24	0.91	2764	939	3435	8	1	16	263	12	495
	52	0.67	0.12	0.89	2525	457	3378	7	2	15	254	26	481
	53	0.72	0.12	0.88	2740	454	3342	7	1	16	263	22	493
	54	0.68	0.17	0.91	2598	628	3462	7	1	15	261	13	484
50	55	0.66	0.09	0.93	2494	343	3497	8	1	16	164	20	299
	56	0.68	0.17	0.93	2538	640	3490	8	1	16	163	11	299
	57	0.68	0.07	0.93	2556	246	3521	8	3	16	166	19	298
	58	0.66	0.12	0.89	2496	455	3349	8	2	16	169	19	300
	59	0.67	0.13	0.87	2496	507	3264	8	2	16	165	19	300
	60	0.67	0.06	0.92	2515	231	3490	8	3	16	160	19	298
30	61	0.78	0.32	0.96	2979	1205	3660	8	2	16	344	56	599
	62	0.78	0.15	0.99	2986	576	3760	8	2	16	327	25	600
	63	0.77	0.22	0.99	2954	829	3769	8	2	16	333	50	598

Table 2: Characteristics of the generated instances, part II.

5.2 Implementation details

We have implemented the mathematical formulations by using the integer linear programming solver CPLEX 11.1. When reporting the experiments with the two formulations, $\mathscr{F}_1$ and $\mathscr{F}_2$, implemented in CPLEX we denote their results by the name of the formulation, i.e. $\mathscr{F}_1$ indicates the formulation as well as its CPLEX implementation. We have not tweaked the CPLEX parameters. We have only set the stopping rule for CPLEX to 3% of the integer solution gap. The computational experiments were executed on a computer equipped with two Xeon 3 GHz processors and 4 GB of RAM.

The metaheuristic algorithm was coded in C++. We have executed the tests for a maximum number of η iterations, and we report the results for different values of this parameter: $\eta = 50 \times 10^3$ and $\eta = 200 \times 10^3$. The ALNS input parameters were determined by some testing on a subset of the instance set. In our tests we have found that good values for the σ parameters are $\sigma_1 = 2, \sigma_2 = 0.1$, and $\sigma_3 = 0.01$. We have used randomly generated integer values for ω in the interval $[\min\{5, |K| \times 0.2]\}, ..., \min\{12, |K| \times 0.8\}]$, and the randomization parameter p was set equal to 0.3. Regarding the master level simulated annealing parameters, we have used a τ_{start} value such that a new solution with objective function value differing by 0.5% from the initial feasible solution will be accepted with a probability of 50%. The cooling rate c was set in such a way that the temperature value at the final iteration is equal to a given value (1000 in our experiments).

We denote by ALNS the algorithm with the same objective function as formulations $\mathscr{F}_1$ and $\mathscr{F}_2$, and by ALNS-RH the modified version with the objective function (6). The γ_1 parameter for ALNS-RH was set in such a way that the second term of the objective function (6) is one order of magnitude larger than the first term.

The metaheuristic algorithm with the objective function of the formulations $\mathscr{F}_1$ and $\mathscr{F}_2$, and the additional term (7) is referred to as ALNS-PS for its peak shaving capabilities. We will report an example of using this algorithm with the parameter γ_2 set equal to one.

5.3 Results

We report in Table 3 the comparison between the formulations $\mathscr{F}_1$ and $\mathscr{F}_2$. The four listed instances are the only ones out of the 63 instances where the formulation $\mathscr{F}_1$ does not exceed memory limits. For these instances we report the objective function values at the best found solutions, as well as the gaps. Here and in the following tables the (percent) gaps are computed as $100 \times (\text{upper bound} - \text{lower bound})/\text{upper bound}$. The formulation $\mathscr{F}_2$ clearly outperforms $\mathscr{F}_1$ and it will therefore be exclusively used in the following experiments.

Tables 4 and 5 compare $\mathscr{F}_2$ and ALNS. We list the lower bounds obtained by $\mathscr{F}_2$ at the end of the computation, the computational times, and the gaps. The metaheuristic algorithm obtains high quality solutions within short computational times, whereas CPLEX can be rather slow on some instances. Using a larger number of iterations for ALNS ($\eta = 50 \times 10^3$

versus $\eta = 200 \times 10^3$) improves solution quality only slightly. This is why in the following experiments we have used $\eta = 50 \times 10^3$. Formulation $\mathscr{F}_2$ is rather useful in assessing the quality of the metaheuristic because it provides good lower bounds. However, we already mentioned that the devised solution approach requires an iterative use of the algorithm for different data assumptions, and for the evaluation of different objective functions. Tables 6 and 7 assess the effect of the rolling horizon. Here formulation $\mathscr{F}_2$ and the metaheuristic algorithms solve smaller instances with a planning horizon of one week. The assignments of groups arriving on the first day of the planning horizon are then fixed, and a new instance is derived as long as the end of the planning horizon does not coincide with the end of the period of 31 days. The lower bound for this type of problem is the same of the one computed by assuming the full month knowledge. Therefore, the solution quality is measured in terms of the gap with respect to this lower bound. For consistency, the reported gap for ALNS-RH considers as upper bound the value of the first term of the objective function only. Because of the rolling horizon framework $\mathscr{F}_2$, ALNS, and ALNS-RH present some infeasibility issues. Both $\mathscr{F}_2$ and ALNS fail in eight instances. The modified objective function of ALNS-RH allows a significant reduction in the number of infeasible solutions which are now only three. However, this advantage comes at the expense of a noticeable worsening of the handling times. We observe that the handling times resulting from the rolling horizon are on average four percentage points worse than those obtained when the instance is solved with the full 31 days horizon. This could be defined as the price to pay for the limited knowledge about the future. When we use ALNS-RH we have an additional eight percentage points worsening which could be seen as the price of the more prudent assumptions of ALNS-RH. Whether the more costly, but more resilient yard plans obtained by ALNS-RH are valuable is a decision to be left to planners' judgment.

Figure 6 illustrates the flexibility given by the objective function term (7) in the ALNS-PS algorithm. The dashed line represents the handling profile induced by the yard assignment decisions computed by the ALNS algorithm for instance 13. This handling profile has a very high peak of 450,000. By imposing a desired largest handling value of 350,000 in the ALNS-PS algorithm we obtain the profile represented in Figure 6 by a solid line. This avoids the high handling peak. However, the yard plan computed by ALNS-PS has a total handling cost 7.6 percentage points higher than the one computed by ALNS.

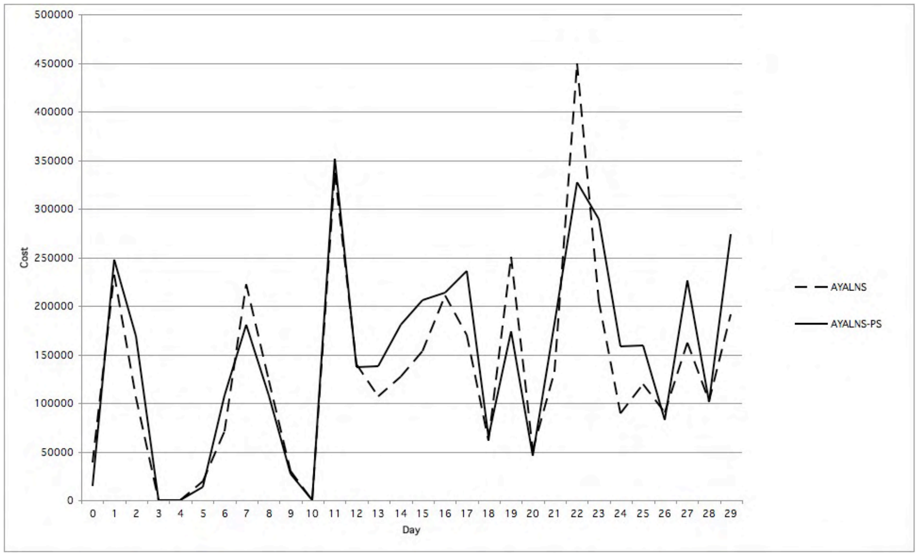

Figure 6: Comparison of handling profiles between ALNS and ALNS-PS algorithms

6 Conclusion

We have described, formulated and solved a yard management problem arising in an automotive transshipment terminal. Several constraints and objectives resulting from managerial rules and policies were considered. We have devised an efficient adaptive large neighborhood search metaheuristic for the problem. Extensive computational experiments clearly show that the proposed metaheuristic yields high quality solutions when benchmarked with a state-of-the-art integer linear programming solver. Furthermore, the metaheuristic can easily handle application specific practical issues such as a rolling horizon, and a manpower leveling objective. A solution approach based on the iterative use of this fast metaheuristic algorithm is then possible for the application under study.

Acknowledgements

This work was partly supported by the MUR (Italy) under project AUTOMA, and by the Natural Sciences and Engineering Council of Canada under grants 227837-09 and 39682-10. This support is gratefully acknowledged. The authors thank the BLG Italia managers Francesco De Bonis and Francesco Scalise for their kind availability in explaining the application, and Alessandra De Luca for her help in defining the instance set. Special thanks are due to Manlio Gaudioso who initiated the cooperation framework that made this research possible.

		$\mathscr{F}_1$			$\mathscr{F}_2$		
$\lvert K \rvert$	Instance index	Objective value	Time (sec)	Gap (%)	Objective value	Time (sec)	Gap (%)
20	3	5179419	798	2.7	5164749	27	2.3
	7	4054548	457	1.8	3987629	12	0.0
	10	4113687	429	2.0	4082843	27	0.5
	42	4678907	1366	0.1	4791606	62	2.4
	Average		763	1.6		32	1.3

Table 3: Comparison between the formulations $\mathscr{F}_1$ and $\mathscr{F}_2$ using CPLEX as integer linear programming solver. The gaps are computed as $100 \times (\text{upper bound} - \text{lower bound})/\text{upper bound}$.

$\lvert K \rvert$	Instance index	$\mathscr{F}_2$ Lower bound	$\mathscr{F}_2$ Time (sec)	$\mathscr{F}_2$ Gap (%)	ALNS $\eta = 50 \times 10^3$ Time (sec)	ALNS $\eta = 50 \times 10^3$ Gap (%)	ALNS $\eta = 200 \times 10^3$ Time (sec)	ALNS $\eta = 200 \times 10^3$ Gap (%)
20	1	3992465	106	2.9	30	0.4	163	0.3
	2	4682893	75	0.4	30	0.8	159	0.8
	3	5043894	27	2.3	32	0.6	172	0.5
	4	4840143	108	2.7	30	0.8	165	0.8
	5	4453641	209	3.0	32	1.6	172	1.3
	6	4928722	112	0.9	32	0.5	172	0.6
	7	3987629	12	0.0	32	0.2	189	0.0
	8	3734106	13	1.6	40	0.1	175	0.1
	9	4710034	17	0.0	36	0.1	175	0.1
	10	4060795	27	0.5	49	0.0	180	0.0
	11	4368846	13	1.6	51	0.6	181	0.3
	12	5064482	13	0.0	48	0.1	194	0.1
40	13	4739218	454	2.4	32	1.5	242	1.4
	14	4525567	755	2.9	33	2.0	234	1.6
	15	5143900	1410	2.5	32	1.3	219	1.1
	16	4662518	306	1.5	33	1.0	244	1.0
	17	4722832	1316	0.9	32	1.6	233	1.1
	18	4705636	1812	1.0	32	2.3	225	1.7
	19	4109508	327	2.1	51	1.8	267	1.0
	20	4057705	860	1.9	54	1.9	268	1.7
	21	4379150	301	0.5	52	1.0	264	0.6
	22	4507207	467	1.0	53	2.5	291	0.6
	23	4241235	643	2.7	55	2.3	299	1.5
	24	4094234	341	0.9	50	1.1	247	0.9
50	25	3560322	844	2.6	48	2.0	255	2.2
	26	3674739	2959	1.6	51	1.3	283	1.2
	27	3752604	2847	0.5	52	1.4	278	0.9
	28	3448997	2865	3.0	52	2.1	260	2.3
	29	3279645	411	2.9	53	2.5	260	1.1
	30	3505008	820	2.3	50	2.5	284	1.6
	Average		682	1.6	42	1.3	225	1.0

Table 4: Computational results, part I. The gaps are computed as $100 \times (\text{upper bound} - \text{lower bound})/\text{upper bound}$.

$\|K\|$	Instance index	$\mathscr{F}_2$ Lower bound	$\mathscr{F}_2$ Time (sec)	$\mathscr{F}_2$ Gap (%)	ALNS $\eta = 50 \times 10^3$ Time (sec)	ALNS $\eta = 50 \times 10^3$ Gap (%)	ALNS $\eta = 200 \times 10^3$ Time (sec)	ALNS $\eta = 200 \times 10^3$ Gap (%)
20	31	5812495	22	1.6	34	0.4	202	0.4
	32	4777588	167	0.8	29	0.5	142	0.5
	33	5609889	711	2.0	30	0.7	130	0.7
	34	5101031	30	0.0	28	2.2	142	0.2
	35	5457213	183	2.4	31	1.4	160	1.2
	36	4870599	203	1.1	34	1.8	159	1.3
	37	5243680	41	1.6	49	0.4	181	0.4
	38	4840143	107	2.7	45	0.8	166	0.8
	39	4515430	61	2.2	46	2.4	158	0.9
	40	5560969	28	2.8	53	0.7	194	0.7
	41	5424514	491	0.8	48	0.4	165	0.3
	42	4678045	62	2.4	50	0.4	175	0.3
40	43	5477806	1123	2.0	29	4.0	174	3.8
	44	5118623	5290	2.6	29	1.9	200	1.7
	45	5284799	7811	1.5	29	1.5	217	1.5
	46	5035190	14052	2.9	31	3.0	227	2.5
	47	5399131	3686	2.9	29	2.9	176	2.0
	48	4881591	18963	1.2	28	5.0	169	4.0
	49	4739218	455	2.4	46	1.5	242	1.4
	50	4594082	9790	1.7	41	4.1	184	2.9
	51	5284799	7832	1.5	43	1.5	218	1.5
	52	4851810	377	2.4	42	3.3	191	2.1
	53	5035190	14942	2.9	46	3.0	226	2.5
	54	5046526	5645	2.8	43	2.0	217	1.4
50	55	3932682	10380	2.4	41	3.9	193	1.8
	56	3995910	510	0.5	44	1.6	216	1.6
	57	4084447	6736	1.3	43	2.0	182	1.9
	58	4030744	817	2.8	47	2.1	220	1.4
	59	3962657	7768	1.3	47	2.3	228	1.4
	60	3868591	6787	1.3	42	3.6	187	2.3
30	61	5225164	690	2.1	43	4.0	156	2.2
	62	5105719	38357	2.6	45	4.5	127	4.3
	63	5270800	559	0.4	43	3.7	129	3.0
	Average		4990	1.9	40	2.2	183	1.7

Table 5: Computational results, part II. The gaps are computed as $100 \times (\text{upper bound} - \text{lower bound})/\text{upper bound}$.

	Rolling horizon		
	$\mathscr{F}_2$	ALNS	ALNS-RH
Instance index	Gap (%)	Gap (%)	Gap (%)
1	7.2	4.3	14.8
2	2.9	1.8	8.8
3	4.8	2.0	11.8
4	n.a.	4.0	5.5
5	6.1	5.9	5.2
6	2.9	3.8	9.9
7	1.7	0.1	13.5
8	3.8	0.4	23.0
9	2.6	1.7	18.3
10	12.4	10.0	16.2
11	4.7	5.0	16.2
12	6.3	4.9	12.6
13	7.0	6.7	17.6
14	7.2	4.6	13.4
15	3.9	6.2	10.9
16	10.6	9.2	13.6
17	n.a.	n.a.	n.a.
18	6.5	6.2	12.6
19	5.1	4.6	11.3
20	6.0	8.2	19.5
21	7.2	3.1	17.1
22	6.8	2.6	20.1
23	7.3	6.8	13.5
24	5.8	7.4	16.8
25	10.2	5.3	14.2
26	9.6	4.1	17.3
27	6.2	5.8	14.6
28	7.8	5.2	13.9
29	11.7	5.8	12.0
30	13.7	6.7	14.7
Average	6.7	4.9	14.1

Table 6: Rolling horizon results, part I. The gaps are computed as $100 \times (\text{upper bound} - \text{lower bound})/\text{upper bound}$. We indicate by "n.a." whenever the algorithm fails in obtaining a feasible solution.

	Rolling horizon		
	$\mathscr{F}_2$	ALNS	ALNS-RH
Instance index	Gap (%)	Gap (%)	Gap (%)
31	n.a.	n.a.	7.0
32	5.5	n.a.	9.7
33	n.a.	n.a.	12.3
34	14.7	14.1	8.2
35	7.5	3.5	8.7
36	7.0	3.6	9.7
37	7.7	n.a.	5.4
38	n.a.	4.0	5.5
39	2.8	7.1	8.3
40	4.7	4.0	7.4
41	n.a.	n.a.	n.a.
42	1.8	3.5	4.9
43	4.7	2.1	16.2
44	7.9	5.7	11.6
45	7.4	3.9	16.3
46	6.4	6.0	12.7
47	8.4	5.3	13.7
48	6.9	7.4	12.2
49	7.0	6.7	17.6
50	8.9	10.0	13.5
51	7.4	3.9	16.3
52	9.4	7.2	12.3
53	6.4	6.0	12.7
54	11.4	6.9	16.0
55	7.8	8.7	18.0
56	7.1	5.4	9.1
57	7.6	6.1	15.3
58	10.5	7.7	16.3
59	7.9	6.3	13.5
60	6.0	7.0	15.4
61	5.5	8.8	11.4
62	n.a.	n.a.	n.a.
63	n.a.	n.a.	8.9
Average	7.3	6.2	13.5

Table 7: Rolling horizon results, part II. The gaps are computed as $100 \times (\text{upper bound} - \text{lower bound})/\text{upper bound}$. We indicate by “n.a.” whenever the algorithm fails in obtaining a feasible solution.

References

Bierwirth, C. and Meisel, F. (2010). A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627.

Cordeau, J.-F., Laporte, G., Legato, P., and Moccia, L. (2005). Models and tabu search heuristics for the berth-allocation problem. *Transportation Science*, 39(4):526–538.

Giallombardo, G., Moccia, L., Salani, M., and Vacca, I. (2010). Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232–245.

Goetschalckx, M. and Ratliff, D. H. (1990). Shared storage policies based on the duration stay of unit loads. *Management Science*, 36(9):1120–1132.

Imahori, S., Yagiura, M., and Ibaraki, T. (2003). Local search algorithms for the rectangle packing problem with general spatial costs. *Mathematical Programming*, 97(3):543–569.

Imahori, S., Yagiura, M., and Ibaraki, T. (2005). Improved local search algorithms for the rectangle packing problem with general spatial costs. *European Journal of Operational Research*, 167(1):48–67.

Lodi, A., Martello, S., and Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252.

Martello, S. and Toth, P. (1992). Generalized assignment problems. In *Algorithms and Computation*, volume 650 of *Lecture Notes in Computer Science*, pages 351–369. Springer, Berlin.

Mattfeld, D. and Orth, H. (2007). The allocation of storage space for transshipment in vehicle distribution. In Kim, K. H. and Günther, H.-O., editors, *Container Terminals and Cargo Systems Design, Operations Management, and Logistics Control Issues*, pages 267–289. Springer, Berlin.

Mattfeld, D. C. (2006). *The Management of Transshipment Terminals, Decision Support for Terminal Operations in Finished Vehicle Supply Chains*, volume 34 of *Operations Research Computer Science Interfaces Series*. Springer, Berlin.

Mattfeld, D. C. and Kopfer, H. (2003). Terminal operations management in vehicle transshipment. *Transportation Research Part A: Policy and Practice*, 37(5):435–452.

Meisel, F. and Bierwirth, C. (2009). Heuristics for the integration of crane productivity in the berth allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):196 – 209.

Moccia, L., Cordeau, J.-F., Monaco, M. F., and Sammarra, M. (2009). A column generation heuristic for a dynamic generalized assignment problem. *Computers & Operations Research*, 36(9):2670 – 2681.

Monaco, M. F., Moccia, L., and Sammarra, M. (2009). Operations research for the management of a transhipment container terminal. The Gioia Tauro case. *Maritime Economics & Logistics*, 11(1):7–35.

Park, Y. M. and Kim, K. H. (2003). A scheduling method for berth and quay cranes. *OR Spectrum*, 25(1):1–23.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

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

Steenken, D., Voß, S., and Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26(1):3–49.

Vis, I. F. A. and Koster, R. D. (2003). Transshipment of containers at a container terminal: An overview. *European Journal of Operational Research*, 147(1):1–16.

Won, S. and Kim, K. (2009). An integrated framework for various operation plans in container terminals. *Polish Maritime Research*, 16(3):51–61.