Expert Systems with Applications 39 (2012) 8337–8349

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Pre-Marshalling Problem: Heuristic solution method and instances generator

Christopher Expósito-Izquierdo, Belén Melián-Batista, Marcos Moreno-Vega *

Universidad de La Laguna, Dpto. de Estadística, IO y Computación, 38271 La Laguna, Spain

ARTICLE INFO

Keywords:
Container terminal
Pre-Marshalling Problem
Heuristics
Instances generator

ABSTRACT

The Pre-Marshalling Problem consists in reshuffling containers in a port yard taking into account that a container with high priority cannot be placed below a container with low priority. The objective of the problem is to minimize the number of movements required to arrange all the containers so that further relocations are not necessary. In this work a heuristic solution method to solve the Pre-Marshalling Problem that significantly outperforms other methods from the literature is proposed. Moreover, an instances generator for this problem with which instances with varying degrees of difficulty can be created is developed. In order to obtain instances with degrees of difficulty that range from low difficulty up to high difficulty, two features that consider both the occupancy rate of the bay of containers and the percentage of containers with high priority that are located below containers with low priority are considered. The computational experiments carried out in this work corroborate the good performance of both the heuristic and the instances generator.

1. Introduction

Over the last decades, containerization has changed the way of general cargo transportation in a drastic way. Since its standardization in the 60's, the container has become the most important element within the global supply chains. Container enables an international end-to-end transportation of goods in a continuous flow from production sources to end customers without interruption.

The steady growth in demand for goods has led to a notorious increase in the number of containers handled worldwide. To face this increased flow in supply chains, governments have opted for the creation of great container terminals capable of managing a large number of containers as a strategic objective. Container terminals are important infrastructures that allow linking between the different means of freight transportation and, at the same time, are outstanding economic engines for new businesses in the regions where they are located (Coto-Millán, Mateo-Mantecón, & Castro, 2010). Competitiveness in this area is huge, so that the main criteria used by the operators to choose a port as operations base are geographic location, politic and social stability, and operational costs (Wiegmans, Rietveld, & Nijkamp, 2001).

In container terminals there is a large number of logistic processes concerning the management of containers forming a complex system to arrange and operate. Usually, container terminals consist of two large container flow interfaces between different means of transportation. On one hand, a container terminal must serve container vessels to perform the tasks of loading and unloading containers and, on the other hand, it must meet the pick-up and delivery operations by the landside means of transportation, typically trucks and trains.

Logistic operations in a container terminal begin with the berthing of vessels coming to port, where it is necessary to provide them a berth and a set of quay cranes to perform the processes of loading and unloading containers. Inbound containers are transported to the container yard, where they are temporarily stored for later collection. Normally, the containers stored in the container terminal are collected by some landside mean of transportation. These means of transportation can also drop off containers in the container terminal and therefore need to be properly handled.

The container yard is one of the most critical points in the management of a container terminal. It is a temporary container storage element for distribution through several means of transportation of goods. Usually, the container yard is found split into a set of container blocks separated by traffic lanes, which are bays of containers arranged in parallel. Every bay has a limited number of container stacks and a maximum stacking tiers. Stacking and retrieval operations to/from container blocks on the yard area are carried out by gantry cranes. The most common types of gantry cranes are rail-mounted gantry cranes (RMGCs), rubber-tired gantry cranes (RTGCs) and overhead bridge cranes (OBCs).

In most cases, containers stored on the container yard are requested by one of the landside means of transportation or are included as outbound containers in the stowage plan of a particular container vessel. Therefore, it is possible to establish a priority for each one of the containers according to their order of

* Corresponding author.
E-mail addresses: cexposit@ull.es (C. Expósito-Izquierdo), mbmelian@ull.es (B. Melián-Batista), jmmoreno@ull.es (M. Moreno-Vega).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2012.01.187

removal. In this sense, the container retrieval order for every bay is known, which allows to determine the movements to do by the corresponding gantry crane to complete the pickup operations.

The organizational structure of the containers on the yard area is a serious constraint to productivity of the container terminal. Containers in every stack are only accessible following a LIFO (Last In, First Out) policy. That is, a container in a low tier can be picked up only if each container on its top is previously retrieved. Although the placement of the containers is carefully planned, a bay could have some high priority containers placed below containers with a low priority. The random arrival of export containers and the lack of accurate information are among the main causes that give rise to misplaced containers into the blocks. This type of situation leads to the performing of relocation movements to gain access to those containers with high priority and consequently produces the presence of delays in container vessel departures and unnecessary waiting times in landside means of transportation.

In many cases, the retrieval order of the containers for every bay is known in time to carry out the relocation of them. Therefore, it facilitates that the containers are always accessible without having to do new relocation movements. In the literature this problem is called Pre-Marshalling Problem (PMP). It can be seen as the problem of reshuffling an initial container bay configuration in order to obtain a bay in which the retrieval operations can be carried out without further relocations, while minimizing the total number of movements required to reshuffle.

With the purpose of solving the Pre-Marshalling Problem, this work presents the Lowest Priority First Heuristic (LPFH) that tries to progressively place containers with lower priorities at the bottom of the stacks or over other already well-located containers using the fewest possible movements. Well-located containers have to be understood as those containers that are not above higher priority containers.

To solve a particular problem, the researcher must first decide the technique to be used (exact technique, heuristic or metaheuristic procedure, approximate algorithm, etc.), and then set the parameters governing the behaviour of the chosen technique. The characteristics of the problem, the previous behaviour of the techniques run over similar problems and the experience of the researcher can help to make the decision. However, in general, in order to decide which technique (or parameters) to use, a large and expensive computational experience has to be developed. The first stage of any computational experiment consists in selecting the experimental units (problem instances). Ideally, instances with different characteristics should be considered (small, medium and large size instances; randomly generated and real application instances; instances of varying degrees of difficulty; etc.). This is not always possible. In general, generating instances of a problem by controlling their degrees of difficulty is not a simple task.

In this work, the design of an instances generator for the Pre-Marshalling Problem is discussed. With the purpose of developing an instances generator for the Pre-Marshalling Problem with which instances of different degrees of complexity can be generated, some intuitive rules that relate the complexity of an instance with the location of the containers in the bay are used. These rules allow to modulate the degree of difficulty of the generated instances.

Finally, the proposed heuristic, LPFH, embedded in a multi-start procedure is used to empirically test the characteristics of the developed instances generator.

In the following, the outline of the work and its highlighted contributions are presented. The Pre-Marshalling Problem description is provided in Section 2. The literature review of the Pre-Marshalling Problem is presented in Section 3. The first contribution of the paper relies upon the development of a heuristic to solve the Pre-Marshalling Problem that significantly outperforms other algorithms from the literature. The description of this heuristic is presented in Section 4. An illustrative example of a problem instance resolution is thoroughly described in Section 5. The second contribution is the development of an instances generator for the Pre-Marshalling Problem that is able to generate instances with varying difficulty degrees. The instances generator is introduced in Section 6. The computational experience carried out in this paper is summarized in Section 7. Finally, the conclusions are highlighted in Section 8.

2. Pre-Marshalling Problem

The Pre-Marshalling Problem is a container relocation problem presented in the container yard and which can cause serious inconveniences to the general operation of the container terminal if is not treated in an adequate way. In maritime container terminals, the outbound containers are stored ones over the others in the port yard following configurations consisting of stacks with various tiers that are called container bays. Containers remain stacked in the bays until they are loaded onto the corresponding container vessel or collected by one of the landside means of transportation. In any case, it is possible to determine the retrieval order of the containers placed in every bay and therefore set a priority to each one representing the order in which it has to be removed.

Container priorities can reflect different scenarios on a container yard. On one hand, it is possible to represent a strict retrieval order in which all container priorities are different from each other. In this type of retrieval orders each container has a clearly defined position within the removing process. One example of this situation is the arrival of trucks at port. On the other hand, it usually exists some retrieval orders in which containers are grouped taking into account a particular criteria and consequently several containers with the same priority can be present in a container bay. This situation is common when the containers are included in a stowage plan or are stacked in line with their weights.

Access to containers that are not at the top of the stacks leads to the need to relocate those containers that are above firstly. This situation can be avoided if the containers to be removed are always at the top of the stacks. That is, it is not frequent to find the containers stacked in the order established by the retrieval plan, so that it is necessary to perform relocations during their removing. These operations delay the departure of container vessels or landside means of transportation and have a negative effect on the efficiency of the terminal.

Fig. 1 shows a bay with 4 stacks, 4 tiers and 9 containers. The priority of each container is represented by a number that indicates the order in which it has to be retrieved. Thus, containers

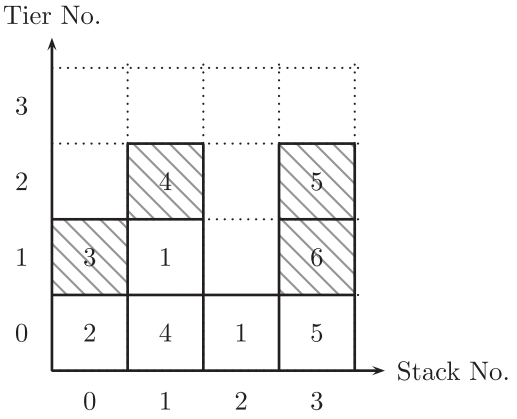

Fig. 1. An example of a bay with 4 stacks, 4 tiers and 9 containers.

with priority 1 have to be retrieved first, then those with priority 2, 3, and so on. Note that, in order to retrieve the containers following their priority orders, relocation operations have to be carried out (see striped containers).

Normally, the container retrieval order is known with several hours in advance. The crane can perform the necessary container relocations so that they are placed following the removal order. The Pre-Marshalling Problem is the problem of reshuffling the containers in a bay with the purpose of obtaining a new container bay layout in which there is not any container with a given priority placed below other container with lower priority. The objective is to minimize the number of crane movements.

A solution to the problem is a sequence of movements with the form (a,b), where a is the origin stack and b is the destination stack of the moved container. Fig. 2 shows a sequence of 7 movements that transforms the initial configuration (on the left hand side of the figure) into a valid container bay layout where there is no a misplaced container (on the right hand side of the figure). In this case, the movement sequence is (2,0), (3,0), (3,2), (1,2), (0,3), (0,1), (0,2).

This work uses the following assumptions presented in Lee and Hsu (2007) for the resolution of the Pre-Marshalling Problem:

1. The problem is restricted to only one container bay. That is, intra-bay movements are not considered because these are very time-consuming in practical situations.
2. All containers in the container bay have the same dimensions. This is the most common situation in container terminals. The placement of containers with different sizes in the same container bay significantly hinders the management of those and, at the same time, leads to the need to take additional security measurements to ensure the integrity of the machinery, workers and freights.
3. Container priorities are known in advance. Each container has a well-defined priority that represents its retrieval order.

3. Literature review

In recent years, container terminals have acquired increasingly a place of privilege within the global economy. Undoubtedly, in this field there is a huge interest in operation research methods to improve the competitiveness of these infrastructures. The number of publications concerning international maritime freight transportation has increased in a remarkable way. Extensive descriptions of the most relevant logistics processes in container terminals and their featured methods of optimization have been done by Stahlbock and Voß (2008), Steenken, Voß, and Stahlbock (2004), and Vis and de Koster (2003).

The number of logistical problems that arise within the container yard is very high indeed. These problems have a very diverse character, from the management of the container yard up to the deployment of horizontal means of transportation. Container relocation problems have increased interest within the field of computer science as it is reflected by the growth of the number of papers published about this issue. Some of the previous works that address the relocation of containers deserve to be highlighted.

A detailed analysis concerning the post-stacking problems for increasing the efficiency of loading/unloading container vessels is performed by Caserta, Schwarze, and Voß (2011). It provides a good summary of the main approaches to solve this kind of problems.

Kim and Bae (1998) propose a methodology to obtain a desirable container bay layout from a current yard map by means of the fewest possible container movements and try to minimize the total travel distance of the corresponding transfer cranes. In their approach, the problem is decomposed into three sub-problems: bay matching, move planning and task sequencing, and solved by mathematical programming techniques.

Kim, Park, and Ryu (2000) formulate a dynamic programming model for the proper location of a new arriving export container taking into account its weight and with the objective of minimizing the number of reshuffle movements at the container vessel loading. Real-time decisions can be made by a decision tree that uses the optimal solution obtained with the mathematical schema developed.

Kang, Ryu, and Kim (2006) propose a simulated annealing search to reduce the number of relocation movements of export containers with inaccurate weight information at the time of loading. In this work, a reduction on the number of relocations is obtained by applying a learning classifier.

Nowadays, there are only a few previous works concerning to the Pre-Marshalling Problem with the same objective as the one considered in this work. That is, minimizing the number of movements in the working sequence of the corresponding crane.

Lee and Hsu (2007) develope an integer programming model based on a multi-commodity network flow. In the structure of the corresponding network, nodes represent slots with containers in the bay, arcs represent container movement possibilities and flows correspond with the container movements between stacks. The work discusses three ways to extend the model to satisfy other requirements in the final layout apart from the number of movements minimization. In any case, computational time is very large, so that the authors propose some strategies to reduce the complexity of the model.

Lee and Chao (2009) propose a neighbourhood search based on a set of subroutines. The heuristic obtains appropriate final layouts for the container bay and then tries to improve the number of movements done by means of an integer programming model. To shorten the movement sequence achieved several improvement techniques are discussed: emptying stacks, removing redundant movements and reducing the mis-overlay index.

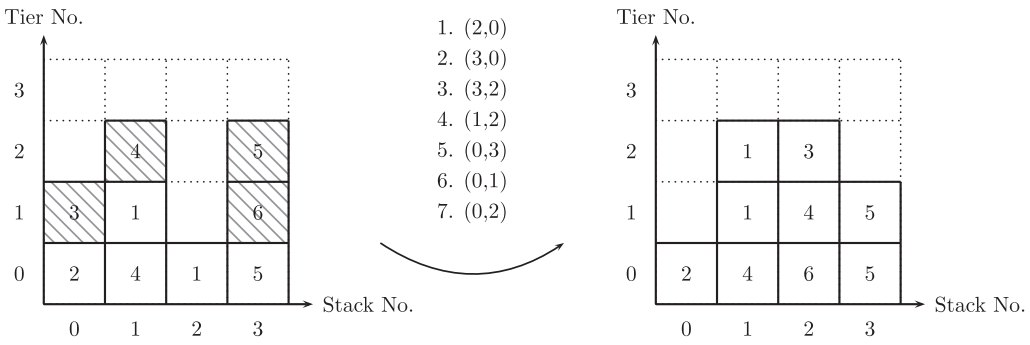

Fig. 2. Example of movement sequence for a bay.

Caserta and Voß (2009) provide a metaheuristic approach based on the paradigm of the Corridor Method. The algorithm builds a corridor with respect to the current solution to determine the destination stack of a specific misplaced container selected by a roulette-wheel mechanism with the information related to the number of forced relocations in each stack of the bay. This work also proposes a local search improvement to intensificate the obtained container bay configuration.

4. Lowest Priority First Heuristic

This section thoroughly describes the Lowest Priority First Heuristic (LPFH) proposed in the work for solving the Pre-Marshalling Problem. The main idea underlying LPFH is that if the containers with the lowest priorities are placed at the bottom of the stacks, then they will not have to be relocated in the future because they do not constitute non-located containers. In this work, a non-located container refers to a container that is above one or more containers with higher priority or above any other non-located containers. Moreover, it is said that a container is well-located in a bay if it has not to be moved to obtain a valid configuration. This happens when it is placed at the bottom of a stack or above containers with lower priority. In any valid configuration of the bay, all containers are well-located.

The Lowest Priority First Heuristic iteratively places each container either at the bottom of a stack or over containers with lower priorities. This is done in reverse order of priorities, starting from those containers with the lowest priority to those ones with the highest priority. That is, if the set of priorities is $P = \{1,\ldots,p\}$, misplaced containers with priority p are relocated in the first place, then containers with priority $(p-1)$, and so on until all containers are well-located. In each iteration, the heuristic determines the container to be relocated, the destination stack and the movements to be performed. The container is then placed on the destination stack at the maximum tier in which it is correctly located. Finally, an improvement process to achieve a reduction in the number of misplaced containers is applied.

Given a container c, let $s(c)$ and $t(c)$ be the stack and the tier at which the container is located, respectively. Let $p(c)$ be the priority of container c and $h(s)$ the number of containers placed on the stack s. Whenever the context so warrants, the container c will be referred to as the pair $(s(c),t(c))$. Schematically, the heuristic is described as follows:

1. $i = p$
2. Repeat
   (a) Let A_i be the set of non-located containers with priority i
   (b) While $(A_i \neq \emptyset)$
      (i) Select a container $c^* \in A_i$
      (ii) Select a destination stack s^*
      (iii) Move container c^* from its current position to stack s^*
   (c) Fill the destination stacks
   (d) $i := i-1$
3. until $(i = 0)$

The number of movements required to properly locate a container c in a stack s^* from its current location in the stack $s(c)$ is stated as follows

$$w(c,s^*) = \begin{cases} f(c,s^*) + g(c,s) + 1, & \text{if } s^* \neq s(c) \\ f(c,s^*) + 1, & \text{if } s^* = s(c) \end{cases} \quad (1)$$

where $f(c,s^*)$ is the minimum number of containers to be removed from the stack s^* to let c be well-located and $g(c,s) = h(s(c)) - t(c) - 1$ is the number of containers placed above c. Note that $f(c,s^*)$ is constant for all the containers with the same priority.

For example, in Fig. 1, in order to properly locate container $(3,1)$ in stack 1, $w((3,1),1) = 5$ movements have to be performed (3 movements to completely empty stack 1, 1 movement to relocate the container with priority 5 placed over the container $(3,1)$ and 1 more movement to place it in the stack 1). However, in order to correctly place the same container in stack 3, $w((4,5),3) = 4$ movements have to be performed.

4.1. Selecting the target container

Let A_i be the set of non-located containers with priority i. That is, each container with priority i that is above other one with a priority lower than i.

$$A_i = \{c : p(c) = i \wedge c \text{ non} - \text{located}\}$$

Regardless of the destination stack, the containers in A_i that require less movements to be properly located have lower values of $g(c,s)$. Thus, the container to be well-located, c^*, is randomly chosen from the λ_1 (parameter set by the user) containers in A_i with the lowest values of $g(c,s)$. Container c^* is called *target container*.

4.2. Selecting the destination stack

The destination stack, s^*, is randomly chosen among the λ_2 (parameter set by the user) stacks with the lowest values of $f(c^*,s)$. Note that the destination stack s^* can be the same as the original stack of the container c^*. That is, it might happen that $s^* = s(c^*)$.

4.3. Moving the target container

Suppose that container c^* has to be moved from its current location in stack $s(c^*)$ to stack s^* (it will be placed at the maximum tier, t^*, at which it is correctly located). In the following explanation $s^* \neq s(c^*)$ is assumed.

Let O_{c^*} be the set of containers placed above c^* in stack $s(c^*)$ and M_{c^*,s^*} be the set of containers in stack s^* located at a tier upper than or equal to t^*. That is,

$$O_{c^*} = \{c : (s(c) = s(c^*)) \wedge (t(c) > t(c^*))\}$$
$$M_{c^*,s^*} = \{c : (s(c) = s^*) \wedge (t(c) \geqslant t^*)\}$$

Note that, before moving container c^* from $s(c^*)$ to s^*, it is necessary to move containers in O_{c^*} and $M_{c^*s^*}$ to other stacks. This is orderly done as explained below.

- Let $o \in O_{c^*}$ and $m \in M_{c^*,s^*}$ be such that

$$t(o) > t(o'), \quad \forall\, o' \in O_{c^*}$$
$$t(m) > t(m'), \quad \forall\, m' \in M_{c^*,s^*}$$

and

$$c' = \begin{cases} o, & \text{if } p(o) < p(m) \\ m, & \text{if } p(o) > p(m) \end{cases}$$

Container c' is selected at random if o and m have the same priority, $p(o) = p(m)$.

- Let $s'(s' \neq s(c^*), s' \neq s^*)$ be a stack with at least an empty slot randomly selected among the λ_3 (parameter set by the user) stacks with the lowest values of $p_{min}(s')$, where $p_{min}(s')$ is the lowest priority of the non-located containers in a stack s'. Empty stacks and stacks without misplaced containers have the highest preference in the selection criteria.
- Move container c' to stack s' and update O_{c^*} and M_{c^*,s^*}.

These steps are repeated while O_{c^*} or M_{c^*,s^*} have some container. Since the containers are considered in reverse order of priority, acting in this way, the number of containers to be moved is minimized in successive iterations.

In the case in which the origin and destination stacks are equal ($s^* = s(c^*)$), only those containers placed at a tier higher than or equal to t^* have to be relocated to other stacks. All these containers, but the target one, are relocated following the procedure explained above for the case in which $s^* \neq s(c^*)$. The target container has to be moved to a temporal stack in order to relocate those non-located containers located beneath it. With the purpose of minimizing the disabled slots in the container bay, the stack with the fewest free slots is selected as temporal one. If there is more than one stack, one of them is randomly selected. Once all non-located containers have been relocated to other stacks, it is possible to move the target container to its destination stack.

4.4. Stack filling

After relocating all containers with a specific priority, an improvement process aimed to reduce the number of non-located containers in the whole container bay is applied. The objective of this phase is to obtain stacks in which all containers are well-located and where the use of the slots is maximized.

When a target container c^* is relocated in a destination stack s^*, this container is at the top of stack s^*. At the same time, note that the destination stack contains only well-located containers. If there are several containers with the same priority, it is possible that some destination stack cannot be filled. This scenario is presented when some destination stack is used to host displaced containers from the subsequent relocation of a container with the priority in question.

The stack filling reasoning is based on trying to take advantage of the empty slots in the destination stack filling them with non-located containers in the most adequate sorted sequence. For each destination stack s^*, those non-located containers with a priority equal to or higher than the container at the top of s^* are candidates to be moved to this stack. Iteratively, containers from lower to higher priority difference with respect to the container at the top of s^* are selected. If there are several non-located containers with the same priority difference, one of them is selected at random. The process ends when all valid destination stacks are full or there are not available non-located containers.

4.5. Multistart procedure

The heuristics described in this section make use of several parameters that confer a degree of randomness to the decisions taken regarding the performed movements. Therefore, if the heuristics were run several times, the obtained solutions would probably be different. Then, the implementation of the previous heuristics is done including them into a multistart procedure, in which they are executed for a given number of iterations.

The stopping criteria used for the completion of the algorithm execution is based on a maximum number of iterations (α) or a certain number of iterations without improvement (β) on the number of movements achieved. In this sense, the stopping criteria can be represented as (α, β).

5. Illustrative example

In order to make this paper self-contained, in the following an example of resolution of a Pre-Marshalling Problem instance using the LPFH is described. Let us consider the container bay previously presented in Fig. 1.

In this example, the bay has capacity for 16 containers (4 stacks $\times$ 4 tiers). There are 9 randomly located containers with priorities ranging from 1, which corresponds to the highest priority, up to 6, which corresponds to the lowest priority. Let us consider the following values for the parameters involved in the previously described heuristic: $\lambda_1 = 1$, $\lambda_2 = 2$ and $\lambda_3 = 3$.

First of all, one container among the λ_1 non-located containers with the lowest priority (6 in this case) is randomly selected. In this example there is a single container with priority 6, located in position (3,1). The number of movements required to relocate this container in the different stacks according to the expression (1) is summarized in Table 1.

A stack among the λ_2 best possible destination stacks is selected at random. Suppose that the selected destination stack is the stack 3. Note that since stacks 0 and 3 require the same number of movements, only one of them is randomly chosen as a possible destination for relocating the target container. Stack 3 has been chosen. In this case, the origin and destination stacks for container (3,1) are the same, stack 3. Therefore, it is required to relocate the containers belonging to stack 3 on other stacks before locating container (3,1) at the bottom of the stack.

In the first place, container (3,2) with priority 5 has to be moved to other stack according to the evaluation of the possible destinations, as indicated in Table 2. These evaluations indicate the lowest priorities of the non-located containers in the stacks, as explained in Section 4.3.

Note that stack 3 is not available to move container (3,2) since it is its origin stack. One of the best λ_3 valid stacks is selected at random. In this case, suppose that stack 0 is chosen. Then, the configuration in Fig. 3 is obtained.

The target container with priority 6 has now to be temporarily moved to other stack. With the purpose of disabling the fewest possible slots in the container bay, the stack with at least an empty slot and the highest number of containers is chosen. Therefore, either stack 0 or stack 1 is selected at random. Suppose that the container is moved to stack 1 as temporal storage obtaining the configuration shown in Fig. 4.

The container with priority 5 located in position (3,0) has now to be moved to other stack according to the evaluations presented in Table 3.

In order to move the container (3,0) with priority 5, one of the λ_3 best available stacks is selected at random. In this case, suppose that stack 2 is chosen. Stack 1 is not considered in the selection process because is the temporal stack for the target container and, at the same time, has no empty slots. Finally, the target container with priority 6 is moved back to stack 3 as it is indicated above. Then the configuration in Fig. 5 is obtained.

At this point, the containers with the lowest priority (6) are well-located, so that containers with priority 5 have to be considered. A container among the λ_1 non-located containers with priority 5 is selected at random taking into account the number of containers they have above. In this example, there are two containers with priority 5 located at positions (0,2) and (2,1). Suppose that the container at position (0,2) is considered to be well-placed in the container bay. The number of movements to relocate it in every stack is indicated in Table 4.

Table 1
Number of movements required to relocate container (3,1).

Stack	Number of movements	Destination stack
0	2 + 1 + 1 = 4	
1	3 + 1 + 1 = 5	
2	1 + 1 + 1 = 3	*
3	3 + 1 = 4	*

Table 2
Destination stack for container (3,2).

Stack	Evaluation	New stack
0	3	*
1	4	*
2	1	*
3	Invalid	

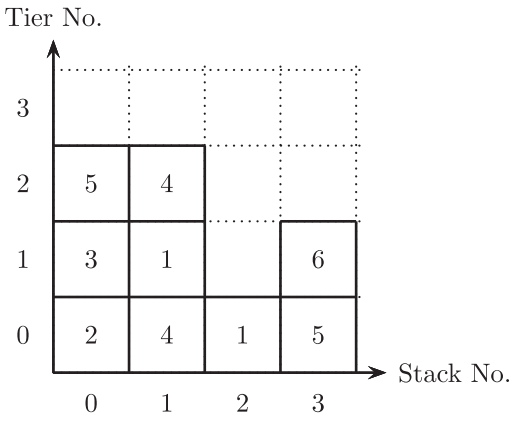

Fig. 3. Container bay layout after movement (3,0).

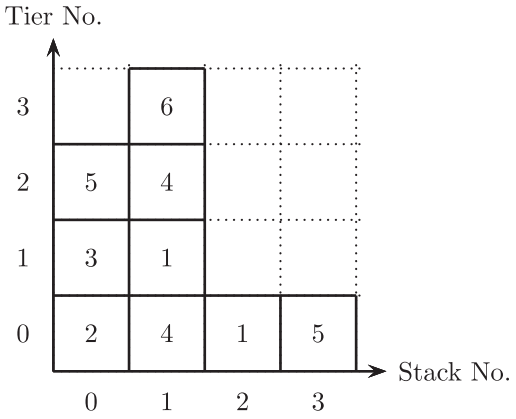

Fig. 4. Container bay layout after movement (3,1).

Table 3
Destination stack for container (3,0).

Stack	Evaluation	New stack
0	5	*
1	Invalid	
2	1	*
3	Invalid	

One stack among the λ_2 best stacks is randomly selected. Suppose that stack 3 is selected as destination stack to locate container (0,2). In this case, there are not containers to be moved in order to relocate the selected target container from position (0,2) to stack 3. Then, the second container with priority 5 located in position (2,1) is considered to be moved according to the evaluations presented in Table 5.

Stack 3 is selected at random as destination for this container. In this case, there are not any additional movements to be performed.

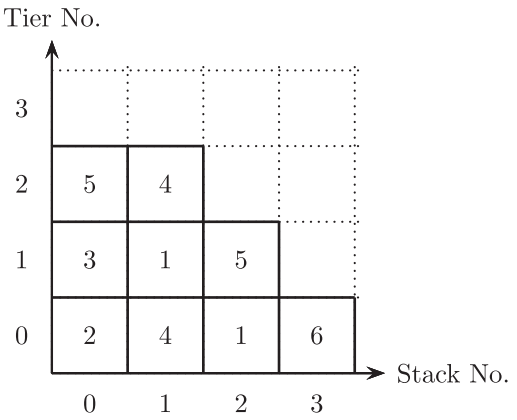

Fig. 5. Container bay layout after movements (3,2) and (1,3).

Table 4
Number of movements required to relocate container (0,2).

Stack	Number of movements	Destination stack
0	3 + 1 = 4	
1	3 + 0 + 1 = 4	
2	2 + 0 + 1 = 3	*
3	0 + 0 + 1 = 1	*

Therefore, the container bay configuration presented in Fig. 6 is obtained.

At this moment, all the containers with priority 5 are well-located, so that the containers with priority 4 have to be considered. A container among the λ_1 non-located containers with priority 4 is then randomly selected. In this case, there are two containers with priority 4, which are located in positions (1,0) and (1,2). However, only container (1,2) has to be relocated because container (1,0) is at the bottom of a stack. Thus, the evaluation of the possible destination stacks for container (1,2) is made, as shown in Table 6.

Suppose that stack 2 is randomly selected as destination among the λ_2 best possible stacks. Therefore, the container with priority 1 located in position (2,0) has to be moved to other stack since its priority is higher than 4. It cannot be moved to neither stack 1, since it is the origin stack of the container (1,2) with priority 4 that has to be moved, nor stack 2, since it is the origin stack of the container (2,0) with priority 1. Therefore, stacks 0 and 3 are available to move the container (2,0) with priority 1. Suppose that stack 3 is selected as destination stack for this container. Then, container (1,2) with priority 4 is moved to stack 2. It can be appreciated that, at this point, only container (0,1) with priority 3 requires to be relocated. In order to do this, a stack among the λ_2 best available stacks is randomly selected. Suppose that stack 2 is chosen in this case, so that container (0,1) with priority 3 is moved without additional required movements. Then, the configuration in Fig. 7 is finally obtained.

At this point, there is not any container requiring relocations, so that the heuristic has obtained a valid configuration performing the movements summarized in Table 7.

Table 5
Number of movements required to relocate container (2,1).

Stack	Number of movements	Destination stack
0	2 + 0 + 1 = 3	
1	3 + 0 + 1 = 4	
2	2 + 1 = 3	*
3	0 + 0 + 1 = 1	*

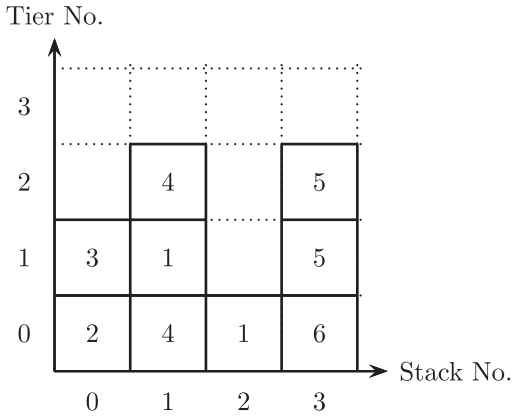

Fig. 6. Container bay layout after movements (0,3) and (2,3).

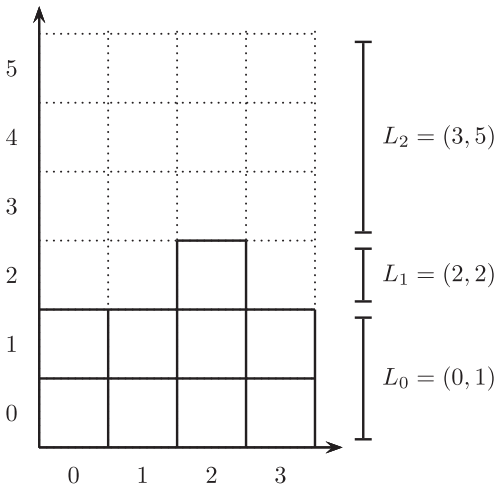

Fig. 8. Levels.

Table 6
Number of movements required to relocate container (1,2).

Stack	Number of movements	Destination stack
0	2 + 0 + 1 = 3	
1	2 + 1 = 3	
2	1 + 0 + 1 = 2	*
3	0 + 0 + 1 = 1	*

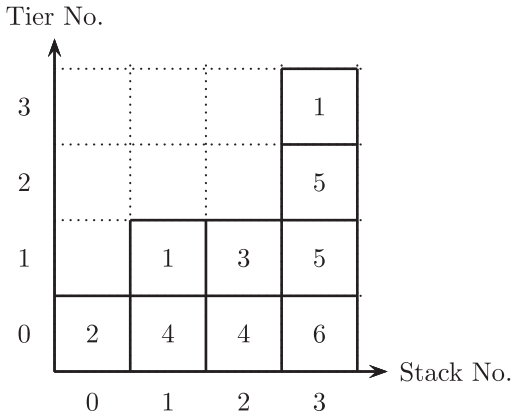

Fig. 7. Container bay layout after movements (2,3) (1,2) and (0,2).

Table 7
Sequence of movements to achieve a valid configuration.

Move	Container	Origin	Destination
1	5	3	0
2	6	3	1
3	5	3	2
4	6	1	3
5	5	0	3
6	5	2	3
7	1	2	3
8	4	1	2
9	3	0	2

6. Instances generator

The proposed instances generator[1] for the Pre-Marshalling Problem is based on two concepts: priority groups and levels in the bay. A priority group consists of a subset of all the possible priorities and a level is a subset of consecutive tiers in the bay. A level L is represented by a pair (l^0, l^1), where l^0 and l^1 are the lowest and the highest of its tiers, respectively. For example, if the set of priorities is $P = \{1,2,3,4,5,6,7\}$ and the container bay has 6 tiers, the priority groups $G_0 = \{1,2,3\}$, $G_1 = \{4,5\}$ and $G_2 = \{6,7\}$, and the levels $L_0 = (0,1)$, $L_1 = (2,2)$ and $L_2 = (3,5)$ can be considered. Fig. 8 shows these levels.

Intuitively, an instance of the Pre-Marshalling Problem is difficult to solve when the percentage of high-priority containers placed below low-priority containers is high. Moreover, the lower the level occupied by high-priority containers, the more difficult the problem is. Another factor that influences the complexity of the instance is the occupancy rate of the bay, defined as the ratio between the number of containers and the number of slots in the container bay. Instances with higher occupation percentage are more difficult to solve. Therefore, it is possible to generate instances with different degrees of difficulty by properly setting the priority groups, the levels of stacking tiers, the percentage of containers of each group located at each level and the occupancy rate of the container bay.

Every instance is then generated using a constructive procedure in which the containers that have to be located in a level are assigned to a feasible stack of that level selected at random. A stack is feasible for a level $L = (l^0, l^1)$ if the last occupied tier of the stack is lower than l^1. Fig. 8 shows that stack 2 is not feasible for level L_1, since its last occupied tier is 2. However, it is feasible for level L_2.

In order to generate an instance of the Pre-Marshalling Problem is necessary to introduce the following parameters:

- S, number of stacks in the container bay,
- T, number of tiers in the container bay,
- q, occupancy percentage of the container bay,
- $P = \{1,2,\ldots,p\}$, set of priorities,
- q_k, percentage of containers with priority $k (k = 1,\ldots,p)$,
- $G_0,\ldots,G_{m-1}$, priority groups,
- $L_0,\ldots,L_{v-1}$, levels,

[1] The instances generator is an open-source software developed in Java Standar Edition version 6. The executable, source code, documentation as well as the user manual can be downloaded from the websitehttps://sites.google.com/site/gciports.

- p_{ij}, percentage of containers of the group i to be located in level $j (i = 0,\ldots,m-1, j = 0,\ldots,v-1)$.

Once the previous parameters have been fixed, the instance is generated as follows:

1. Calculate the number of containers of the instance, n, as a function of the number of stacks and tiers of the container bay and its occupancy percentage as follows.

$$n = \lceil q \cdot S \cdot T \rceil.$$

2. Obtain the number of containers with priority k, n_k, as follows.

$$n_k = q_k \cdot n \quad \forall k = 1,\ldots,p \tag{2}$$

It has to be satisfied that

$$\sum_{k=1}^{p} n_k = n$$

and

$$n_k \geqslant 1 \quad \forall k = 1,\ldots,p$$

3. Calculate the number of containers of each priority group to be placed in each bay level. This number is obtained as follows.

$$n_{ij} = p_{ij} \sum_{k \in G_i} n_k, \quad \forall i = 0,\ldots,m-1 \quad \forall j = 0,\ldots,v-1 \tag{3}$$

Note that sometimes it will be necessary to adjust the value of some n_{ij} to ensure that all containers of group G_i are located in some level. That is, it must be satisfied that

$$\sum_{j=0}^{v-1} n_{ij} = \sum_{k \in G_i} n_k, \quad \forall i = 0,\ldots,m-1$$

4. For each level j, construct the set CL_j made up of containers to be placed in this level by randomly selecting n_{ij} containers of each group G_i. The location of the containers in the bay is done level by level as follows:
   (a) Set $j = 0$
   (b) Repeat
   i Obtain the set FS_j consisting of feasible stacks for level $L_j = \left(l_j^0, l_j^1\right)$
   ii While $((CL_j \neq \emptyset)$ and $(FS_j \neq \emptyset))$
   A. Select at random a container from CL_j and a stack from FS_j
   B. Place the container in the selected stack
   C. Update the set FS_j
   D. Update the set CL_j

   iii if $(FS_j = \emptyset)$
   A. Randomly place the containers in CL_j on tiers $l_j^1 + 1, l_j^1 + 2, \ldots$ orderly completing each level. That is, a container cannot be placed at a tier if there are available slots in the previous tier

   iv $j = j + 1$

   (c) until $(j = v)$

6.1. Example

In order to illustrate the behaviour of the developed instance generator when obtaining an instance for the Pre-Marshalling Problem, the following data are considered. A container bay with $S = 4$ stacks, $T = 6$ tiers, $q = 80\%$, $P = \{1,2,\ldots,7\}$, $q_1 = q_2 = q_4 = 20\%$, $q_3 = q_5 = q_6 = q_7 = 10\%$, $G_0 = \{1,2,3\}$, $G_1 = \{4,5\}$, $G_2 = \{6,7\}$, $L_0 = (0,1)$, $L_1 = (2,2)$, $L_2 = (3,5)$, $p_{00} = 50\%$, $p_{01} = 20\%$, $p_{02} = 30\%$, $p_{10} = 20\%$, $p_{11} = 50\%$, $p_{12} = 30\%$, $p_{20} = 20\%$, $p_{21} = 50\%$, $p_{22} = 30\%$.

The first step is to determine the number of containers of each priority, n_k, and the number of containers of each priority group that have to be placed in each bay level, n_{ij}, as indicated by the expressions (2) and (3), respectively. Then, for each level j, the set of containers CL_j that have to be placed at this level by randomly selecting n_{ij} containers of the group G_i $(i = 0,\ldots,m-1)$ is constructed. These values are summarized in Table 8.

At level 0, all the stacks are feasible, $FS_0 = \{0,1,2,3\}$, and two containers can be placed at each stack. Since $CL_0 \neq \emptyset$, a container from CL_0 and a stack from FS_0 are randomly selected. The container is then placed in that stack. This step is repeated until all the containers in CL_0 are placed. In the same way, at level 1, all the stacks are feasible, $FS_1 = \{0,1,2,3\}$, and in each stack a container can be placed, except in the stack 1, where two containers can be placed. Since $CL_1 \neq \emptyset$, a container from CL_1 and a stack from FS_1 are randomly selected and the container is placed in the stack. This step is repeated while feasible stacks are available. Since $CL_1 \neq \emptyset$, for each container in CL_1, a container is randomly selected and placed in that stack. Note that the tiers are orderly completed. That is, containers 1 and 4 of CL_1 are located at the tier 3 because there are available positions in this stack. The procedure ends by analogously placing containers in CL_2. Fig. 9 shows the generated instance. Containers in CL_0, CL_1 and CL_2 appear shaded with the same colour. Note that the containers in CL_i are located in level L_i or in one inferior level, whenever this is possible (this happens with containers in CL_1 and CL_2). If it is not possible, they are placed in the lowest possible tiers (containers in CL_1).

7. Computational experiments

The computational experiments carried out pursue several well-defined objectives. In the first place, identifying an appropriate set of parameter values for the execution of the algorithm proposed in this paper. Secondly, checking the performance of the proposed heuristic with respect to other works from the literature. Thirdly, analyzing the contribution of the stack filling method in the quality of the reached solutions. Finally, verifying the relationship between the arrangement of the containers in the bay, which is given by the instances generator, and the difficulty when solving the obtained problem instance with LPFH.

LPFH was programmed by using Java SE 6 language and all computational experiments presented in this section were executed on a PC with an Intel Core 2 Duo E8500 3.16 GHz and 4 GB of RAM memory.

The benchmark sets of instances used in this work were taken from Caserta and Voß (2009). These instances have dimensions $3 \times 3,\ldots,3 \times 8$, $4 \times 4,\ldots,4 \times 7$, $5 \times 4,\ldots,5 \times 10$, 6×6 and 6×10

Table 8
An example of a randomly generated instance.

Groups	Priority	n_k	n_{ij}		
			L_0	L_1	L_2
G_0	1	4	5	2	3
	2	4			
	3	2			
G_1	4	4	1	3	2
	5	2			
G_2	6	2	1	2	1
	7	2			

$CL_0 = \{1,1,2,2,3\} \cup \{4\} \cup \{6\} = \{1,1,2,2,3,4,6\}$
$CL_1 = \{1,2\} \cup \{4,4,5\} \cup \{6,7\} = \{1,2,4,4,5,6,7\}$
$CL_2 = \{1,2,3\} \cup \{4,5\} \cup \{7\} = \{1,2,3,4,5,7\}$

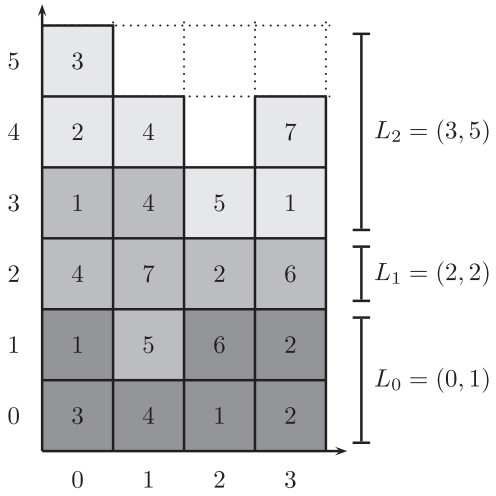

Fig. 9. A randomly generated instance.

(19 instance sets). In each case, the set is composed of a total of 40 instances with ratios of 100% occupancy and, in each one, the containers have different priorities to each other. To allow the implementation of the container relocations, 2 rows of empty slots at the top of the container bays have been considered, as it is done in Caserta and Voß (2009).

7.1. Parameters setting

The combinations of values considered for the parameters λ_1, λ_2 and λ_3 governing the lowest priority first heuristic described in Section 4 are summarized in Table 9, where S indicates the number of stacks.

In order to determine the differences among these combinations of parameters, the Friedman nonparametric statistical test is used (see, for instance, Daniel, 1990). In the cases in which the null hypothesis of equality of treatments is rejected, we use the multiple comparisons test of Friedman to determine the differences among the combinations.

Table 10 summarizes the results obtained when applying the Friedman test for the comparison of equality in the mean number of moves required by the proposed heuristic. The results corroborate that there are significant differences in the mean number of moves when the parameters change. Table 11 shows the results obtained when applying the multiple comparisons test of Friedman at $\alpha = 0.05$ significance level for the mean number of moves. Means with the same letter are not significantly different.

Table 9
Values for the parameters λ_1, λ_2 and λ_3.

λ_1	λ_2	λ_3
1	1	1
1	1	0.25S
1	1	0.5S
1	1	S
1	0.25S	1
1	0.25S	0.25S
1	0.25S	0.5S
1	0.25S	S
1	0.5S	1
1	0.5S	0.25S
1	0.5S	0.5S
1	0.5S	S
1	S	1
1	S	0.25S
1	S	0.5S
1	S	S

Table 10
p-value associated to the statistic test of Friedman.

Variable	Statistic test value	Theoretical value	p-value
Number of moves	88,4326	22,31	$\leqslant$0,1

Table 11
Multiple comparisons of Friedman. $\alpha = 0.05$.

Group	treatment	Ranks
a	$1 - 0.25S - 0.5S$	196.5
a	$1 - 0.25S - S$	186
a	$1 - 1 - S$	180.5
ab	$1 - 1 - 0.5S$	172
ab	$1 - 0.25S - 0.25S$	170
ab	$1 - 0.5S - S$	169.5
ab	$1 - 0.5S - 0.5S$	168.5
ab	$1 - 0.5S - 0.25S$	157
bc	$1 - 1 - 0.25S$	132.5
cd	$1 - S - 0.5S$	113.5
cd	$1 - S - 0.25S$	109.5
cd	$1 - S - S$	105.5
cd	$1 - 0.5S - 1$	104.5
cd	$1 - S - 1$	93
d	$1 - 0.25S - 1$	86.5
e	$1 - 1 - 1$	31

From the obtained results, we conclude that there are not statistically significant differences for several combinations of parameters regarding the mean number of required moves. Therefore, we select the values $\lambda_1 = 1$, $\lambda_2 = 0.25S$ and $\lambda_3 = 0.5S$.

7.2. Comparative analysis with the literature results

In the following, a comparative analysis between the results provided by the proposed heuristic with those reported by the metaheuristic based on the Corridor Method paradigm presented in Caserta and Voß (2009) is performed. The computational study conducted in Caserta and Voß (2009) was restricted only to the sets of instances with sizes 3×3, 4×4, 5×5 and 6×6. For each set of instances, only the first 10 instances were considered.

It is relevant stressing that the computational experiments carried out in Caserta and Voß (2009) were executed on a Pentium IV Linux Workstation with 512 MB of RAM memory and the metaheuristic developed was implemented using the C++ language.

Table 12 presents the comparative analysis between the two resolution techniques for the Pre-Marshalling Problem. The first two columns define the set of instances and the maximum stacking height allowed. Columns three and four specify the performance in terms of number of movements and the execution time of the metaheuristic approach. Columns five and six show the number of movements and the execution time obtained by the LPFH. LPFH was executed with the parameter values $\lambda_1 = 1$, $\lambda_2 = 0.25S$ and $\lambda_3 = 0.5S$ and with a stopping criteria (150, 100), where 150 refers

Table 12
Comparative between Corridor Method and LPFH.

Set	Max. height	Corridor		LPFH	
		Movements	Time	Movements	Time
3×3	5	21.30	20.00	9.20	4.15×10^{-4}
4×4	6	28.27	20.00	17.40	1.47×10^{-3}
5×5	7	46.91	20.00	30.50	3.61×10^{-3}
6×6	8	50.14	20.00	49.60	5.97×10^{-3}

to the maximum number of iterations and 100 to the maximum number of iterations without improvement. Importantly, due to the small size of the 3×3 and 4×4 instance sets, the values of parameters used in these cases are $\lambda_1 = 1$, $\lambda_2 = 2$ and $\lambda_3 = 2$. In every case, the results presented correspond to average values of 10 independent executions of the resolution methods and the execution times are expressed in seconds.

As shown in the data summarized in Table 12, the performance of the proposed heuristic is superior to that of the metaheuristic. The number of movements obtained in each case is smaller than that of its counterpart, specially for sets of instances with small dimensions. Furthermore, in spite of the fact that LPFH was executed on a faster computer, it can be seen that the execution times obtained by LPFH are much lower than those obtained by the corresponding metaheuristic. Although the results can be found in the field of milliseconds, are expressed in seconds for reasons of comparability.

7.3. Comparative analysis with optimal results

In order to make a more detailed and extensive computational analysis, a comparison of the results achieved by means of the use of the heuristic developed with regard to the optimal solutions of the problem is performed.

The optimal solutions were obtained through the implementation of an A* search algorithm. The path-cost function used to explore every node in the search tree is established by the number of movements done into the partial solution and the heuristic used to estimate the distance to a valid configuration of the bay is based on the number of existing non-located containers. It should be noted that the number of non-located containers in the bay is a lower bound for the number of movements to achieve a valid configuration of the Pre-Marshalling Problem.

For each instance, the search was executed for a maximum of one hour of CPU time. Small instances can be solved by the A* search algorithm, which provides the corresponding optimal solutions, within the set time limit. However, large instances cannot be solved using the described search using practical CPU times.

Table 13 shows the comparative analysis carried out between the LPFH and the optimal solutions achieved by the A* search algorithm. The results correspond to the average values of 10 independent executions and the runtimes are expressed in seconds. The stopping criteria was (150, 100), where 150 refers to the maximum number of iterations and 100 to the maximum number of iterations without improvement. The parameter values used in the execution of the heuristic were $\lambda_1 = 1$, $\lambda_2 = 0.25S$ and $\lambda_3 = 0.5S$. Similarly to the previous section, for the sets of instances 3×4, 4×4 and 5×4, the parameter values were $\lambda_1 = 1$, $\lambda_2 = 2$ and $\lambda_3 = 2$ due to the small number of stacks. In the set of instances with size 3×3, since using only 2 empty rows at the top of the container bay leads to that some parameter value combinations with deterministic characteristics are not valid. In this case, the parameter values were $\lambda_1 = 1$, $\lambda_2 = 3$ and $\lambda_3 = 2$. It is necessary to comment that such instances dimensions are not found in real situations. Therefore, it does not invalidate the results achieved in this work.

The presented results demonstrate the good performance and the low computational times of the heuristic developed in this work. The heuristic shows high effectiveness for those sets of instances for which optimal solutions are available. In each case, the average number of movements obtained are close to the optimal values. The increment in the number of movements as the size of the considered instances increases is progressive, which shows that the heuristic presents a good robustness. Moreover, the computational times used in the execution of LPFH are quite reduced (markedly below half a second in all cases) demonstrating its computational efficiency.

Table 13
Comparative between optimal solutions and LPFH.

Set	Max. height	Optimal	LPFH	
			Movements	Time
3×3	5	8.78	10.95	5.76×10^{-4}
3×4	6	9.03	11.03	8.63×10^{-4}
3×5	7	10.15	11.98	1.33×10^{-3}
3×6	8	11.28	13.40	1.77×10^{-3}
3×7	9	12.80	15.40	2.57×10^{-3}
3×8	10	13.68	16.38	2.89×10^{-3}
4×4	6	15.83	20.10	1.66×10^{-3}
4×5	7	21.05	22.13	2.57×10^{-3}
4×6	8	–	24.20	3.32×10^{-3}
4×7	9	–	27.88	4.36×10^{-3}
5×4	6	–	31.18	2.31×10^{-3}
5×5	7	–	31.78	3.93×10^{-3}
5×6	8	–	38.40	4.88×10^{-3}
5×7	9	–	41.43	5.36×10^{-3}
5×8	10	–	47.80	6.90×10^{-3}
5×9	11	–	53.73	9.44×10^{-3}
5×10	12	–	58.08	1.12×10^{-2}
6×6	8	–	51.55	6.57×10^{-3}
6×10	12	–	77.90	1.41×10^{-2}

7.4. Stack filling contribution

The stack filling process presented in Section 4.4 allows to reduce the number of non-located containers in the bay taking advantage of the existing empty slots in a proper way. In the following, an analysis of the contribution of the stack filling process is done. This study is based on a comparison between the quality of the solutions obtained using the stack filling and without using it in the structure of the proposed heuristic.

The characteristics of the executions are similar to those presented in the previous section. Fig. 10 shows a histogram with the number of movements achieved for each one of the available set of instance through the application of LPFH using stack filling and without using it.

As expected, the number of movements conducted during the execution of the heuristic when the stack filling process is used is lower than the scenario in which this process is not used. This situation is exacerbated as the size of the considered instances increases. For example, for the set of instances of size 6×10, the difference in the number of movements reaches nearly 40 units.

Interestingly, for the sets of instances with sizes 3×3 and 3×4 the heuristic performance without the use of stack filling is slightly higher. The justification can be found in that the stack filling process completes the empty slots of stacks in a very fast way using non-located containers in the bay, but for instances with a small size these decisions may cause containers with low priority are not used during the filling and need further relocations.

7.5. LPFH results

It is reasonable to conjecture that the difficulty of an instance of the Pre-Marshalling Problem depends, among other factors, on the occupancy rate of the bay and on the positioning of the containers in the different levels. In order to empirically verify this conjecture, several instances with different characteristics to measure the computational effort required by the LPFH to solve them have been generated.

Instances with $T = 4$ tiers, $S = \{4, 7, 10\}$ stacks and occupancy rates $q = \{50\%, 75\%, 100\%\}$ have been considered. The number of containers of each priority is equal to 4. The priorities have been

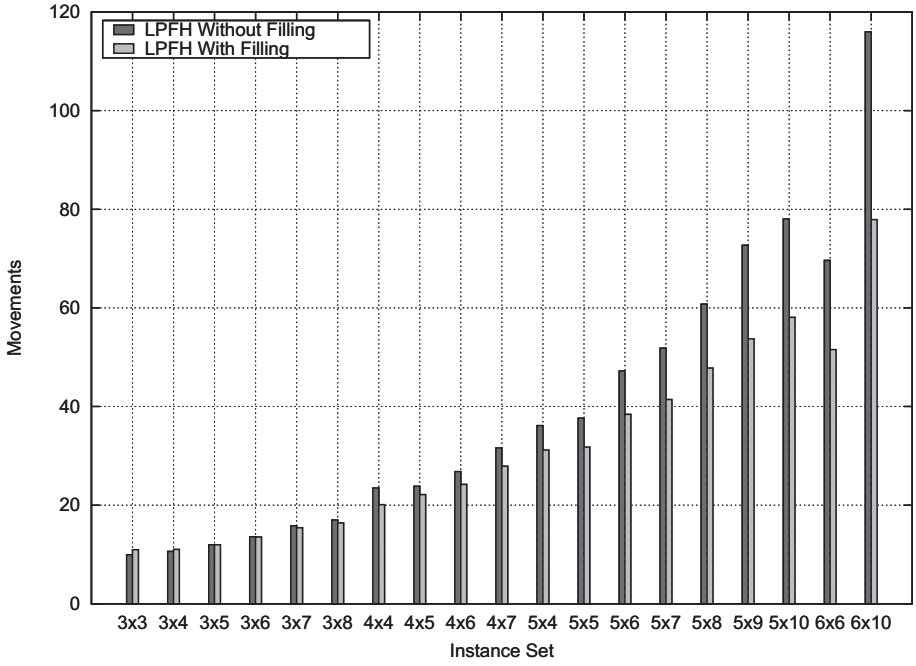

Fig. 10. Histogram of the contribution of the stack filling process in the LPFH.

grouped into three equal sized groups G_0,G_1, and G_2 formed by high, medium and low priorities, respectively. In all cases, the levels of the bay coincide with its tiers. This means that the first level consists of the first tier, the second level consists of the second tier, and so on. Table 14 summarizes the distribution by level of the containers in each group (values p_{ij}). Note that there are three configurations with distinct characteristics. The instances of the first category seem to be more difficult to solve than the instances of the second. The same fact is observed for the instances of the second category with respect to the instances of the third category.

The behaviour of LPFH depends on the values of the parameters λ_1, λ_2 and λ_3. Thus, LPFH behaves almost as a greedy strategy if $\lambda_1 = \lambda_2 = \lambda_3 = 1$. On the other hand, it resembles a random procedure when the values of these parameters are increased. The values used in the experiments carried out in this section depend on the sizes of the instances and are shown in Tables 15–17. In all cases, $\lambda_1 = 1$ is considered.

Once S, T, q and p_{ij} were set, 10 instances were randomly generated. Each instance was solved 5 times using different combinations of (λ_2, λ_3). Let *mov* and $iter_{best}$ be the minimum number of movements required to reach a valid configuration and the iteration at which this value is obtained, respectively. Note that *mov* is the objective function value of the Pre-Marshalling Problem. In Tables 15–17, the results obtained are reported. The first four columns of these tables show the occupancy rate q, the instances configuration and the values of the parameters λ_2 and λ_3, respectively. Then the average of the variables *mov* and $iter_{best}$ is given for each instance category.

As shown in the obtained results, once the occupancy rate q is fixed, the difficulty of the instances is increased by increasing the percentage of containers of each group located over containers of

Table 14
Levels, groups and configurations.

	Levels	Tiers	Groups		
			G_0 (%)	G_1 (%)	G_2 (%)
C_0	L_3	3	0	0	0
	L_2	2	0	0	100
	L_1	1	0	100	0
	L_0	0	100	0	0
C_1	L_3	3	25	25	25
	L_2	2	25	25	25
	L_1	1	25	25	25
	L_0	0	25	25	25
C_2	L_3	3	50	25	0
	L_2	2	50	25	0
	L_1	1	0	25	50
	L_0	0	0	25	50

Table 15
Problems with 4 tiers and 4 stacks.

q (%)	C	λ_2	λ_3	*mov*	$iter_{best}$
50	C_0	1	1	3.10	1.00
		3	3	3.14	11.54
		4	4	3.30	13.96
	C_1	1	1	1.60	1.00
		3	3	1.60	4.48
		4	4	1.62	5.58
	C_2	1	1	1.00	1.00
		3	3	1.00	2.96
		4	4	1.00	4.14
75	C_0	1	1	15.00	1.00
		3	3	15.58	17.52
		4	4	16.02	18.12
	C_1	1	1	8.00	1.00
		3	3	6.78	11.40
		4	4	7.04	14.10
	C_2	1	1	1.90	1.00
		3	3	1.90	2.64
		4	4	1.90	3.08
100	C_0	1	1	30.00	1.00
		3	3	22.88	19.80
		4	4	23.70	18.24
	C_1	1	1	17.33	1.11
		3	3	14.86	18.10
		4	4	14.84	19.50
	C_2	1	1	8.50	1.00
		3	3	8.14	9.74
		4	4	8.07	16.33

Table 16
Problems with 4 tiers and 7 stacks.

q (%)	C	λ_2	λ_3	mov	$iter_{best}$
50	C_0	1	1	7.50	1.00
		2	3	7.30	2.14
		7	7	8.38	9.08
	C_1	1	1	4.60	1.00
		2	3	4.52	1.92
		7	7	4.94	3.66
	C_2	1	1	2.80	1.00
		2	3	2.54	2.24
		7	7	2.80	3.32
75	C_0	1	1	23.20	1.00
		2	3	21.14	11.38
		7	7	22.44	20.36
	C_1	1	1	13.40	1.00
		2	3	12.36	4.24
		7	7	13.84	12.10
	C_2	1	1	9.20	1.00
		2	3	7.84	3.70
		7	7	8.20	6.66
100	C_0	1	1	40.10	1.02
		2	3	35.80	17.80
		7	7	45.66	7.50
	C_1	1	1	28.00	1.06
		2	3	25.62	15.26
		7	7	30.24	17.64
	C_2	1	1	18.20	1.02
		2	3	16.40	10.76
		7	7	18.78	15.56

Table 17
Problems with 4 tiers and 10 stacks.

q (%)	C	λ_2	λ_3	mov	$iter_{best}$
50	C_0	1	1	11.20	1.00
		2	3	10.94	2.68
		10	10	13.62	13.98
	C_1	1	1	7.60	1.00
		2	3	6.92	2.72
		10	10	8.18	9.28
	C_2	1	1	4.30	1.00
		2	3	4.30	1.36
		10	10	4.88	6.46
75	C_0	1	1	31.80	1.00
		2	3	29.38	18.46
		10	10	32.38	11.30
	C_1	1	1	21.00	1.00
		2	3	18.82	11.64
		10	10	21.46	16.96
	C_2	1	1	12.30	1.00
		2	3	11.34	5.76
		10	10	12.54	14.98
100	C_0	1	1	53.70	1.00
		2	3	47.68	9.64
		10	10	64.68	9.98
	C_1	1	1	38.40	1.00
		2	3	36.14	11.30
		10	10	46.22	7.58
	C_2	1	1	27.20	1.00
		2	3	26.46	16.42
		10	10	31.50	11.14

higher priority groups. In all cases, the average number of movements required to obtain a valid configuration for the instances belonging to category C_0 is higher than the required by the instances in category C_1. The same fact is observed for the instances of category C_1 with respect to the instances of category C_2. Moreover, the effort required by LPFH to reach the best valid configuration increases, regardless of the values of the parameters λ_2 and λ_3. This means that once the values of these parameters are fixed (and, therefore, the behaviour of the heuristic), the instances of category C_0 are always more difficult to solve.

Furthermore, the difficulty of the instances also increases by increasing the occupancy rate of the container bay. For each category of instances, C_i, the number of movements required to obtain a valid configuration in instances with an occupancy rate $q = 100\%$ is always higher than in the rest of the instances. The easiest instances to solve are the ones with $q = 50\%$.

Finally, it can be observed that, in general, the best objective function values are obtained when $\lambda_2 = 2$ and $\lambda_3 = 2$. This corresponds to a semi-greedy behaviour of the LPFH.

8. Conclusions

Container yard constitutes one of the most essential elements within a container terminal. The role of the container yard is of vital importance because it brings together the main flows of containers, those existing between container vessels and landside means of transportation. The proper management of containers in the container yard has a direct impact on the productivity of the container terminal. Thus, it is necessary to maintain a good container stacking policy and a rigorous forecast about the delivery of these.

Container relocation problems have a special relevance for the container terminal because these can produce delays on the service time of container vessels or the landside means of transportation if are not managed in a appropriate way. The Pre-Marshalling Problem is one of these problems. Its objective is to reshuffle the containers placed in a container bay with the intention that a container that will be removed shortly can never be found beneath other container with a low priority.

In order to obtain high quality solutions for the Pre-Marshalling Problem the Lowest Priority First Heuristic (LPFH) is proposed. This heuristic is based on placing the misplaced containers with the lowest priorities at the bottom of the stacks or beneath other containers with higher priorities by means of the minimum possible number of movements. The results of the comparative analysis carried out with respect to a previous approach from the literature and an exact approach demonstrate the good performance of the developed heuristic. Moreover, the robustness of LPFH is assessed since it is able to solve varying difficulties and problem sizes without changing the values of its parameters.

Finally, this work proposes an instances generator for the Pre-Marshalling Problem with which it is possible to generate instances of varying degrees of difficulty. Some of the most remarkable factors that impact on the difficulty of the problem are the occupancy ratio of the container bay and the layout of the available containers. Several sets of instances with different degrees of difficulty are proposed to be used as benchmark units to evaluate the performance of resolution techniques for the Pre-Marshalling Problem. The instances generator is available online.

Acknowledgments

This work has been partially funded by the European Regional Development Fund, the Spanish Ministry of Science and Technology (Projects TIN2009-13363 and TIN2008-06872-C04-01), and the Canary Government (Project PI2007/019).

References

Caserta, M., Schwarze, S., & Voß, S. (2011). Container rehandling at maritime container terminals. In J. W. Böse (Ed.), *Handbook of terminal planning. Operations research computer science interfaces series* (Vol. 49, pp. 247–269). New York: Springer.

Caserta, M., & Voß, S. (2009). A corridor method-based algorithm for the pre-marshalling problem. *Lecture Notes in Computer Science, 5484*, 788–797.

Coto-Millán, P., Mateo-Mantecón, I., & Castro, J. V. (2010). The economic impact of ports: Its importance for the region and also the hinterland. In *Essays on port economics. Contributions to economics* (pp. 167–200). HD: Physica-Verlag.

Daniel, W. W. (1990). *Applied nonparametric statistics*. Boston: PWS-Kent Publishing Company.

Kang, J., Ryu, K., & Kim, K. (2006). Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent Manufacturing, 17*, 399–410.

Kim, K. H., & Bae, J. W. (1998). Re-marshaling export containers in port container terminals. *Computers & Industrial Engineering, 35*(3-4), 655–658 [Selected Papers from the 22nd ICC and IE Conference].

Kim, K. H., Park, Y. M., & Ryu, K.-R. (2000). Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research, 124*(1), 89–101.

Lee, Y., & Chao, S.-L. (2009). A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research, 196*(2), 468–475.

Lee, Y., & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers & Operations Research, 34*(11), 3295–3313.

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

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

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

Wiegmans, B.W., Rietveld, P., Nijkamp, P. (2001). Container terminal services and quality. Serie Research Memoranda, 0040.