UNIVERSIDAD POLITÉCNICA DE VALENCIA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

MODELS AND TECHNIQUES FOR PLANNING, OPTIMIZATION AND SIMULATION IN CONTAINER TERMINALS

Master Thesis in
Artificial Intelligence, Pattern Recognition
and Digital Image

February 2011

Supervised by:
Dr. Miguel Á. Salido Gregorio
Dr. Federico Barber Sanchís

Presented by:
Mario Rodríguez Molins

Models and Techniques for Planning, Optimization and Simulation in Container Terminals

Mario Rodríguez Molins

Master Thesis in
Artificial Intelligence, Pattern Recognition
and Digital Image

Departamento de Sistemas Informáticos y Computación

Valencia, 2011

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Maritime Container Terminals

During the last decades, maritime transportation (liner shippings) is being increasingly used to carry products to different countries by companies around the world. Container terminals generally serve as a transshipment on ships and land vehicles (trains or trucks) for these products. These products are stored into containers in order to facilitate their management (loading, unloading or transhipment).

Containers were standardized by the International Organization for Standardization (ISO) based upon the US Department of Defense standards between 1968 and 1970, ensuring interchangeability between different modes of transportation worldwide. The standard sizes and fitting and reinforcement norms that exist now evolved out of a series of compromises among international shipping companies, European railroads, U.S. railroads, and U.S. trucking companies.

Four important ISO recommendations standardised containerisation globally [55]:

- *R-668* in January 1968 defined the terminology, dimensions and ratings;
- *R-790* in July 1968 defined the identification markings;
- *R-1191* in January 1970 made recommendations about corner fittings;
- *R-1897* in October 1970 set out the minimum internal dimensions of general-purpose freight containers.

The term *twenty-feet-equivalent-unit* (TEU) is used to refer to one container with a length of twenty feet. Thereby, a container of 40 feet is expressed by 2 TEU. This measure is also used to identify the capacity of the vessels, e.g. the vessel *EMMA MÆRSK* (built on 2006) is one of the largest vessels which can carry 14770 TEU[1].

[1] Updated on November 2010, `http://www.maerskline.com/link/?page=brochure&path=/our_services/vessels`

Henesey shows in [24] how this transshipment market is growing fast. Figure 1.1 indicates the growth in number of containers handled for Europe and Mediterranean. The total transshipment has increased more than twofold over 1995-2004 and by 58 per cent over 2000-2004 to 22.5 million TEU.

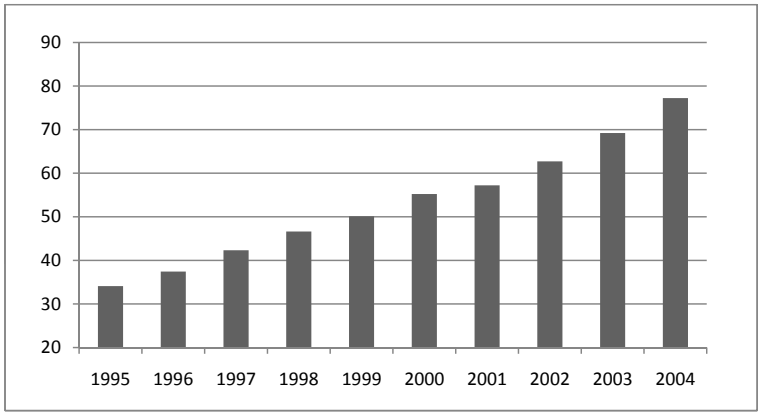


Figure 1.1: N. Europe/S. Europe/Mediterranean Container Throughput (million TEU).

According to review done by Drewry Consultants [10], Global economic trends means that container throughput at the world's port fell for the firs time ever (Figure 1.2), from 524 million TEU in 2008 to 473 million TEU in 2009 (around 10%). However, their forecast is that container throughput will increase globally by an average of 7.2% a year between 2009 and 2015, as the economic recovery takes hold. During this same six-year period, however, those container terminal expansion projects that are considered to be confirmed will increase capacity by an annual average of just 2.9%. The much slower rate of capacity growth relative to throughput will inevitably increase terminal utilisation rates. In 2009 Drewry estimates that around 62.9% of container terminal capacity was being utilised worldwide.

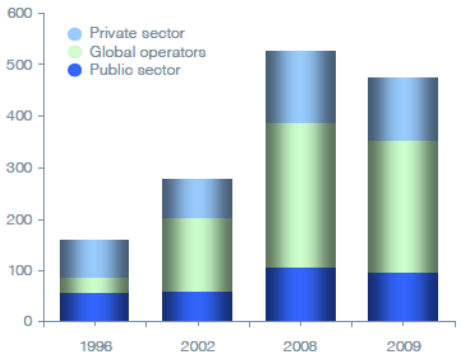
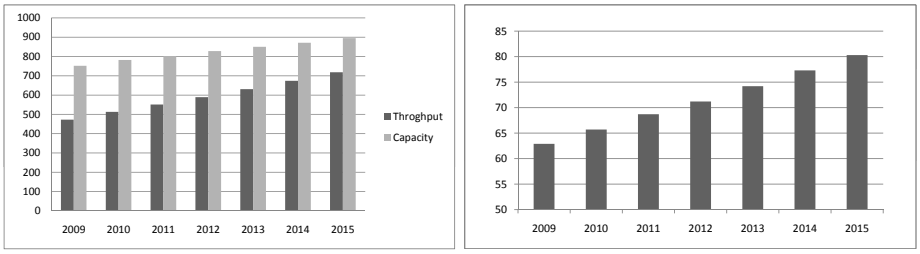


Figure 1.2: Public/private control of container terminals (million TEU).

Between 2009 and 2015, Drewry's forecasting indicates that global container

throughput will rise by a total of 246 million teu, from 473 million to 718 million teu, an increase of just under 52%. The capacity of the world's container terminals is forecast to grow by 143 million teu during the same time frame, a rise of just 19%.



(a) Throughput and Capacity (million TEU) (b) Utilization (%)

Figure 1.3: Forecast supply and demand in the global container port market, 2009-2015 (source [10]).

In November 2009, IHS Global Insight[2], a recognized global leader in economic and financial analysis and forecasting, completed an evaluation of the economic contribution of the liner shipping industry using 2007 as a base year. Among their conclusions, it can be remarked that:

- Cargo transported by the liner shipping industry represents about one-third of the value of total global trade, equating to more than US$ 4.6 trillion worth of goods;
- Liner shipping companies deployed more than 400 services providing regularly scheduled service, usually weekly, connecting all countries of the world.

1.2 Optimization Problems in Container Terminals

Within a Container Terminal there are several handling activities dependent on various related subsystems 1.4. The four main subsystems are [25]:

1. *ship-to-shore* movements to unload the containers from ship to berth (or in reverse order, to load them onto the ship);
2. *transfer* bi-directional movement of containers from berth to stack (storage area), from one stack to another stack and from the hinterland to a stack;
3. *storage* stack or area where containers are placed to wait until their next ship, train or truck; and

[2]http://www.worldshipping.org/benefits-of-liner-shipping/global-economic-engine

4. *delivery/receipt* movement of containers from stack to the hinterland transport, or vice versa.

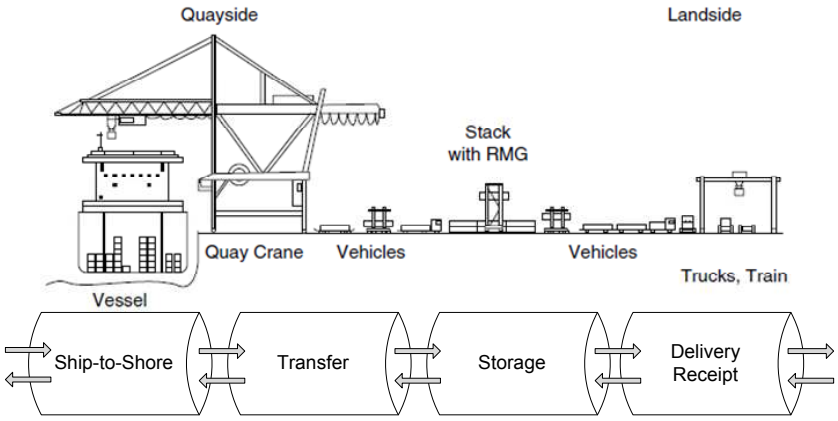


Figure 1.4: Process of unloading and loading a vessel.

Depending on the Container Terminal, these activities will be focused on either transshipment, import or export containers. They are briefly described in [41]. With transshipment, containers are unloaded from a ship onto the storage yard where they are stored for an amount of time. Then, they are loaded on a ship again to continue on their way to their destination. With export, the container is transported from the hinterland to the terminal, where it is stored for an amount of time before being brought to the quay side where it is loaded onto a ship. For import, this process is in the reversed order.

In Figure 1.4, these activities are illustrated following the steps one container does when it is at one of these terminals. The whole process is described in [69]. Once a vessel arrives at the port, a berth has to be assigned to it as well as a number of Quay Cranes (QCs). The import containers have to be taken off from that vessel employing these assigned QCs. They take the containers off the ship's hold or off the deck. Regarding these QCs, they must be manned since automated vehicles do not have enough precission, for instance to position the spreader to pick up one container. QCs leave these containers on vehicles, trailers or automated guided vehicles (AGV) to take them to the stack (storage area). These vehicles are responsible for internal transport within a Container Terminal (transfer subsystem).

This stack allows to store the containers during a certain period. It is composed by a set of block where the containers are distributed in bays, rows and tiers (Figure 1.5). These lanes are managed by transfer cranes, straddle carriers (SCs) or rail-mounted container gantry cranes (RMGs). These equipments maintain different stacking capacity (see Figure 1.6 [32]). A description and a brief comparision of the mentioned handling equipment is done in [25, 66].

Once the containers must be retrieved from the stack, the same vehicles or cranes are used to transfer them to other transportation modes. Import containers

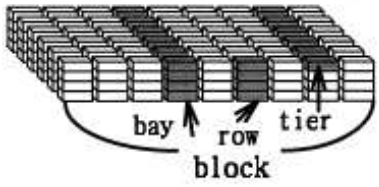


Figure 1.5: Schematic overview of one container block.

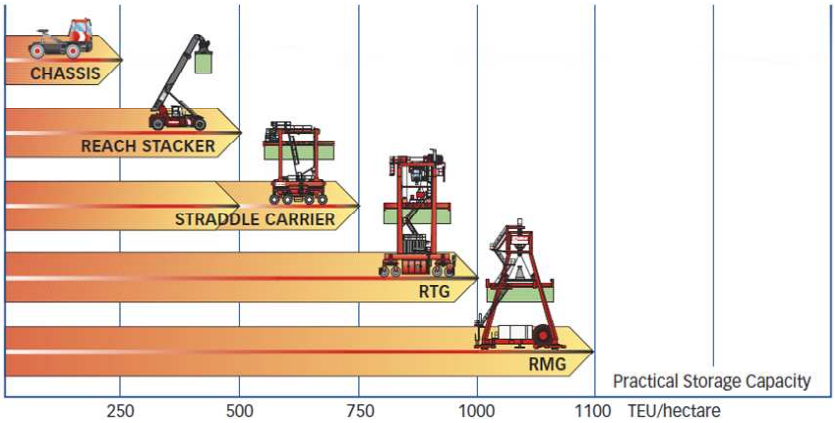


Figure 1.6: Different types of handling equipment and their stacking capacity.

will be transfered to barges, trucks or trains, and transshipment containers onto deep sea vessels.

In case of export containers, this process is executed in reverse order. Thereby, the containers from barges, trucks or trains are loaded onto the vessels.

Each of the activities described above results in different subprocesses (Figure 1.7). Shaded bubbles indicate the different decission problems which must be faced by the Container Terminals. For instance, the Berthing Allocation problem to decide where and when locate a vessel at the quay or the Routing and Scheduling Problem to manage all the automated guided vehicles (AGVs) within a Container Terminal. In [66] (an update of [67]) provide a comprehensive survey of the state of the art of operations at a container terminal as well as the methods for their optimization. More information regarding transshipment operations can be seen in [45, 69]. They distinguish decisions on container handling according to the time horizon involved. A time horizon in decisions for the strategic, tactical, and operational level covers from one to several years, from a day to months, and only a day, respectively.

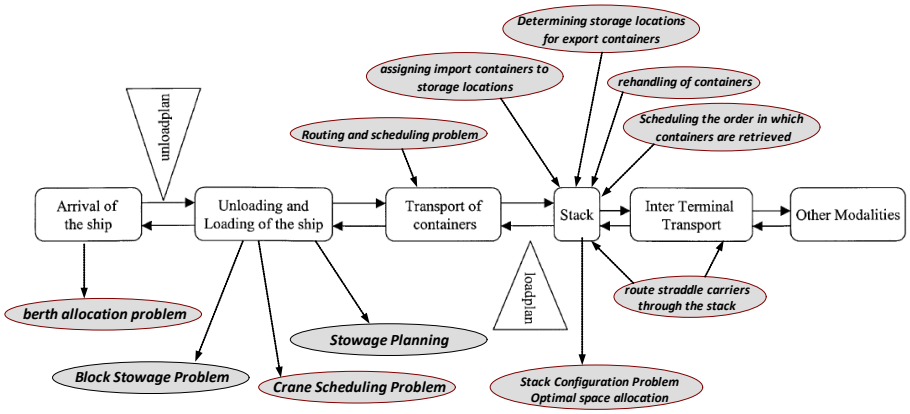


Figure 1.7: Optimization Problems in Container Terminals.

1.3 Motivation

Competition between different Container Terminals around the world makes necessary to handle more and more containers in short time and at low cost. Therefore, Container Terminals are forcen to enlarge handling capacities and strive to achieve gains in productivity. This purpose can be meet by:

1. Designing new terminals with advanced layouts [37], e.g. automatization in particular regions with high labor cots.

2. The replacement of the older equipment with a more efficient one.

3. With the existing infrastructure and equipment, achieving higher profits by means of powerful information technologiy and logistics control software systems including optimization methods.

Another important issue for the success at any container terminal is to forecast container throughput accurately [6]. With this data, they could develop better operational strategies and investment plans.

Focusing on using both infrastructure and equipment efficiently, Operations Research (OR) has been usually employed, it means, techniques like linear programming or mixer-integer programming were employed to optimize the decission problems. However, during the last years, artificial intelligence (AI) techniques are proved to be efficient to manage these problems [19]. These techniques are based on planning (e.g., metricFF [29] or LPG-TD [15] planners using the standard encoding language *PDDL*[16]), scheduling [2, 49], metaheuristics (simulated annealing [39], GRASP [13] or tabu search [18]), neural networks [4], and so on.

Generating efficient solutions for all these problems is necessary in order to avoid accidents like the ones in Figure 1.8. In this accident, the containers were damaged due to the fact that terminal operators did not consider the difference between the heights of the different rows. Deadlocks or idle times are other problems

Figure 1.8: Unbalanced yard-bay at one vessel.

that arise due to poor coordination between the handling equipment. For instance, Figure 1.9 shows several trucks waiting for containers to be unloaded by the Quay Cranes.

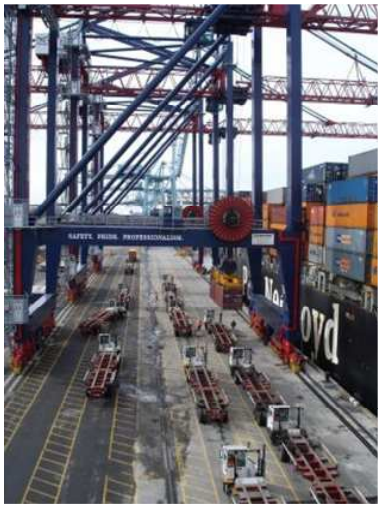


Figure 1.9: Unproductive times loading/unloading vessels.

The overall goal collaboration between our group at the Technical University of Valencia (UPV) and the maritime container terminal MSC (Mediterranean Shipping Company S.A) is to offer assistance to help in planning and scheduling tasks such as the allocation of spaces to outbound containers, to identify bottlenecks, to determine the consequences of changes, to provide support in the resolution of incidents, to provide alternative planning of vessel arrivals, etc.

Given all the decission problem a Container Terminal must face, in this work we will focus our attention on problems related to the storage of the containers into the yard (Housekeeping and Container Stacking problem) in Chapter 2 and the arrival of the vessels (Berthing Allocation and Quay Crane Assignment Problems) in Chapter 3. Next, in Chapter 4 we integrate of these two problems. The main

conclusions of this work as well as some guidelines for future work are presented in Chapter 5. In this chapter, the publications that support this work are presented as well.

Chapter 2

Container Stacking Problem

2.1 Introduction

The container traffic has increased over the last decades. The liner shippings are building larger vessels in order to be able to handle the traffic of millions of containers (TEU) around the world. The operating costs for these vessels are high and it is very important to shorten their turn-around or berthing time in Container Terminals.

Reducing the berthing time implies that the other activities in Container Terminals work efficiently. One of these activities where the time might be reduced is the loading and unloading of containers from/into containers yards.

In Container Terminals, the loading operation for export containers is preplanned. For load planning, a container-ship agent usually transfers a load profile (an outline of a load plan) to a terminal operating company several days before a ship's arrival. The load profile specifies only the container group and not individual containers. In order to have an efficient load sequence, storage layout of export containers must have a good configuration.

Loading and offloading containers on the stacks is performed by cranes following a 'last-in, first-out' (LIFO) storage. In order to access a container which is not at the top of its pile, those above it must be relocated. It occurs since other ships have been unloaded later or containers have been stacked in the wrong order due to lack of accurate information. This reduces the productivity of the cranes. Maximizing the efficiency of this process leads to several requirements:

1. Each incoming container should be allocated a place in the stack which should be free and supported at the time of arrival.

2. Each outgoing container should be easily accessible, and preferably close to its unloading position, at the time of its departure.

In addition, there exist a set of hard/soft constraints regarding the container locations, for example, small differences in height of adjacent yard-bays, dangerous

containers must be allocated separately by maintaining a minimum distance and so on.

Nowadays, the allocation of positions to containers is usually done manually. Therefore, using appropriate Artificial Intelligent techniques is possible to achieve significant improvements of lead times, storage utilization and throughput.

Within Container Terminals, the design of the layout of container yards is an influencial factor in productivity [37]. Figure 2.1 shows a common container yard. Usually, a yard consists of several blocks, and each block consists of 20-30 yard-bays [38]. Each yard-bay contains several (usually 6) rows. Each row has a maximum allowed tier (usually tier 4 or tier 5 for full containers). Figure 2.1 also shows the gantry cranes used to move a container within a stacking area or to another location on the terminal. For safety reasons, it is usually prohibited to move the gantry crane while carrying a container [43], therefore these movements only take place in the same yard-bay.

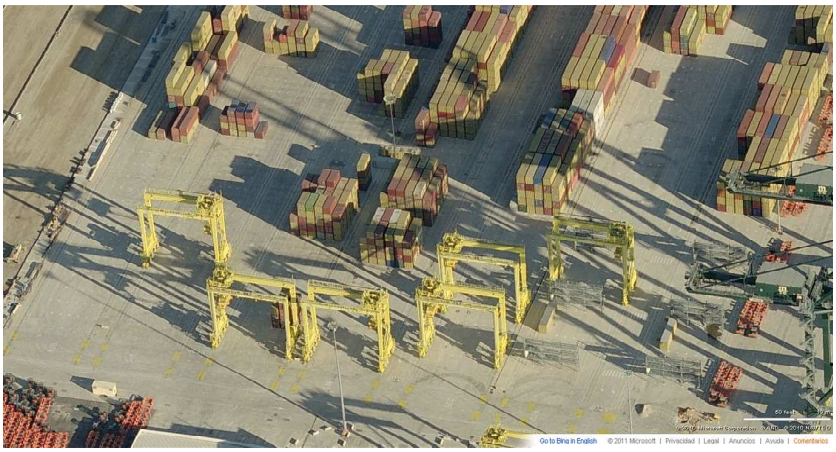


Figure 2.1: A container yard with gantry cranes (MSC Valencia Port).

The main focus of this chapter is to present a planning system which optimally reallocates outgoing containers for the final storage layout from which a load planner can construct an efficient load sequence list. In this way, the objective is therefore to plan the movement of the cranes so as to minimize the number of unnecessary movements (reshuffles) of containers in a complete yard. To this end, the yard is decomposed in yard-bays, so that the problem is distributed into a set of subproblems. Thus, each yard-bay generates a subproblem, but containers of different yard-bays must satisfy a set of constraints among them, so that subproblems will be sequentially solved taken into account the set of constraints with previously solved subproblems.

2.2 Literature Review

Given the Container Stacking Problem, the first issue to tackle is the own designing of the container layout of the Container Terminal. Thus, there are some studies where some methods are proposed methods for designing layouts of container yards [37]. For evaluating this design, parameters as orientation of the blocks to the quay (either parallel or perpendicular) and the number of lanes are taken into consideration.

The Container Stacking Problem can be managed in two different ways according to when it should be done the optimization:

1. minimizing the number of relocations during the pickup (or loading) operation, and

2. getting a desirable layout for the bay before the pickup operation is done in order to minimize (or eliminate) the number of relocations during this process (*remarshalling*).

Dekker et al. [12] explore different stacking policies for containers in automated terminals during the loading process by means of simulation. They distinguish between two strategies. The former one is based on categories (e.g., having the same destination, weight...). Two containers of the same category can be interchanged, and can thus be stacked on top of each other without the risk a lower container in a stack is needed before the ones on top of it have been removed. The latter strategy focuses on the departure times of the containers: a container can only be stacked on top of containers that all have a latter (planned) departure time than the departure time of the container to be stacked.

In [34], authors propose a methodology to estimate the expected number of rehandles to pick up an arbitrary container and the total number of rehandles to pick up all the containers in a bay for a given initial stacking configuration. In a similar way, two methods to minimize the number of relocations during the pickup operation are compared in [36], a branch-and-bound algorithm against a heuristic rule based on an estimator.

Kim and Bae [35] also propose a methodology to convert the current layout into the desirable layout by moving the fewest possible number of containers (remarshalling) and in the shortest possible travel distance. Although it takes a considerable time since they use mathematical programming techniques. Cooperative coevolutionary algorithms have been developed in [47] to obtain a plan for remarshalling in automated container terminals.

Within these automated container terminals, a solution based on simulated annealing [8] rearranges all the containers to be loaded onto the vessels to ensure no reshuffles when pickup operation is performed. This paper uses two non-crossing stacking cranes to build a solution for the sequence of movements.

Following the same trend, Lee and Chao [42] consider only one bay in their model by applying a neighborhood search heuristic. This heuristic is composed by

different subroutines, being one of them a binary integer programming in order to reduce, when it is possible, the needed movements.

Other techniques have been developed by reinforcement learning. Specifically, Q-Learning has been employed to achieve a goal layout given by the shipping order [27, 26].

This chapter is focused on the remarshalling process [33]. We present a domain-dependent planner to solve the remarshalling problem. In this chapter a new heuristic with a set of optimization criteria is introduced. This heuristic achieves efficiency and takes into account constraints that should be considered in real-world problems.

The reminder of this chapter is organized as follows: the next section presents the Container Stacking Problem and how is modeled. Section 2.4 describes the developed planner. Section 2.5 and 2.6 introduce the different optimization criteria added to the planner to be adapted to the real-world requirements. The next section presents the necessary modifications to the planner in order to optimally manage the container blocks. An analytic formula is given in Section 2.8 to estimate the number of reshuffles of a container block. Finally, the last sections show the results as well as conclusions.

2.3 Problem description

The Container Stacking Problem can be viewed as a modification of the *Blocks World* planning domain [70], which is a well-known domain in the planning community. This domain consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The *Blocks World* planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower (or on a table). The optimal *Blocks World* planning problem is to do so in a minimal number of moves.

Blocks World problem is closed to the Container Stacking Problem, but there are some important differences:

- The number of towers is limited to 6 because a yard-bay contains usually 6 rows.
- The height of a tower is also limited to 4 or 5 tiers depending on the employed cranes.
- There exist a set of constraints that involve different rows such as balanced adjacent rows, dangerous containers located in different rows, etc.
- The main difference is in the problem goal specification. In the *Blocks World* domain the goal is to get the blocks arranged in a certain layout, specifying the final position of each block. In the container stacking problem the goal state is not defined as accurately, so many different layouts can be a solution for a problem. The goal is that the most immediate containers to load are in

the top of the towers, without indicating which containers must be in each tower.

We can model our problem by using the standard encoding language for classical planning tasks called *PDDL* (Planning Domain Definition Language) [16]. The purpose is to express the physical properties of the domain under consideration and it can be graphically represented by means of tools as [23]. A classical Artificial Intelligence planning problem can be defined by a tuple $\langle A, I, G \rangle$, where A is a set of actions with preconditions and effects, I is the set of propositions in the initial state, and G is a set of propositions that hold true in any goal state. A solution plan to a problem in this form is a sequence of actions chosen from A that when applied transform the initial state I into a state of which G is a subset.

Following the *PDDL* standard, a planning task is defined by means of two text files. The **domain file**, which contains the common features for problems of this domain. The **problem file** describes the particular characteristics of each problem. These two files will be described in the following subsections.

2.3.1 Domain specification

In this file, we will specify the *objects* which may appear in the domain as well as the relations among them (*propositions*). Moreover, in order to make changes to the world state, *actions* must be defined.

- *Object types*: *containers* and *rows*, where the rows represent the areas in a yard-bay in which a tower or stack of containers can be built.

- *Types of propositions*:

  - Predicate for indicating that the container `?x` is on `?y`, which can be another container or, directly, the floor of a row (stack).

    ```
    on ?x - container ?y - (either row container)
    ```

  - Predicate for indicating that the container `?x` is in the tower built on the row `?r`.

    ```
    at ?x - container ?r - row
    ```

  - Predicate for stating that `?x`, which can be a row or a container, is clear, if there are no containers stacked on it.

    ```
    clear ?x - (either row container)
    ```

  - Predicate for indicating that the crane used to move the containers is not holding any container.

    ```
    crane-empty
    ```

  - Predicate for stating that te crane is holding the container `?x`.

    ```
    holding ?x - container
    ```

  - Predicates used to describe the problem goal. The first one specifies the most immediate containers to load, which must be located on the top of the towers to facilitate the ship loading operation. The second one becomes true when this goal is achieved for the given container.

    `goal-container ?x - container` and

    `ready ?x - container`
  - Numerical predicates. The first one stores the number of containers stacked on a given row and the second one counts the number of container movements carried out in the plan.

    `height ?s - row` and

    `num-moves`

- *Actions*:

  - The crane picks the container `?x` which is in the floor of row `?r`.

    `pick (?x - container ?r - row)`
  - The crane puts the container `?x`, which is holding, in the floor of row `?r`.

    `put (?x - container ?r - row)`
  - The crane unstacks the container `?x`, which is in row `?r`, from the container `?y`.

    `unstack (?x - container ?y - container ?r - row)`
  - The crane stacks the container `?x`, which is currently holding, on container `?y` in the row `?r`.

    `stack (?x - container ?y - container ?r - row)`
  - Finally, we have defined two additional actions that allow to check whether a given (goal) container is ready, that is, it is in a valid position. When a container is clear:

    `fict-check1 (?x - container)`

    The container is under another (goal) container which is in a valid position.

    `fict-check2 (?x - container ?y - container)`

As an example of *PDDL* format, we show in Figure 2.2 the formalization of the stack operator. Preconditions describe the conditions that must hold to apply the action: crane must be holding container `?x`, container `?y` must be clear and at row `?r`, and the number of containers in that row must be less than 4. With this constraint we limit the height of the piles. The effects describe the changes in the world after the execution of the action: container `?x` becomes clear and stacked on `?y` at row `?r`, and the crane is not holding any container. Container `?y` becomes not clear and the number of movements and the containers in `?r` is increased in one unit.

```
(:action stack
 :parameters (?x - container ?y - container ?r - row)
 :precondition (and
    (holding ?x) (clear ?y)
    (at ?y ?r)   (< (height ?r) 4))
 :effect (and
    (clear ?x)   (on ?x ?y)
    (at ?x ?r)   (crane-empty)
    (not (holding ?x))
    (not (ready ?y))
    (not (clear ?y))
    (increase (num-moves) 1)
    (increase (height ?r) 1)))
```

Figure 2.2: Formalization of the *stack* operator in *PDDL*.

This domain specification is characterized by not prioritizing any container. However, the domain can be changed to take into account different container groups depending on their departure times [64]. In Figure 2.3, the different container groups are identified by means of different colors: red, light gray, dark gray and black, respectively. The red ones are the first ones to be loaded into the next vessel and, as it can be observed in the final layout in the figure, they are located on top of the stacks to facilitate their loading.

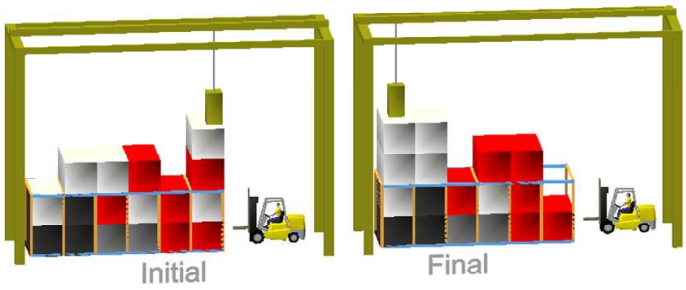


Figure 2.3: Initial state example (left), and final layout achieved (right).

2.3.2 Problem specification

Once the problem domain has been defined, we can define problem instances. These files describe the particular characteristics of each problem:

- *Objects*: the rows available in the yard-bay (usually 6) and the containers stored in them.
- *Initial state*: the initial layout of the containers in the yard.
- *The goal specification*: the selected containers to be allocated at the top of the stacks or under other selected containers.

- *The metric function*: the function to optimize. In our case, we want to minimize the number of relocation movements (reshuffles).

Figure 2.4 shows a simple example of a problem instance in PDDL. This problem describes a scenario with three containers (X1, X2 and X3) in the yard-bay and the crane is not holding any of them. X2 is an export container which must be relocated in order to be easily accessible at loading time of the next vessel.

```
(define (problem p1) (:domain CStackP)

 ;;Domain objects
 (:objects  X1 X2 X3 - container S1 S2 S3 S4 S5 S6 - slot )

 ;;Initial state
 (:init
(clear X1)(clear X3)(clear S1)(clear S4)(clear S5)(clear S6)
(on X2 S2)(on X1 X2)(on X3 S3)(at X1 S2)(at X2 S2)(at X3 S3)
(= (height S1) 0)(= (height S2) 2)(= (height S3) 1)
(= (height S4) 0)(= (height S5) 0)(= (height S6) 0)
(= (num-moves) 0)(handempty)
(goal-container X2)
 )

 ;;Goal state
 (:goal (and (handempty) (ready X2)) )

 ;;function to optimize
 (:metric minimize (num-moves))
)
```

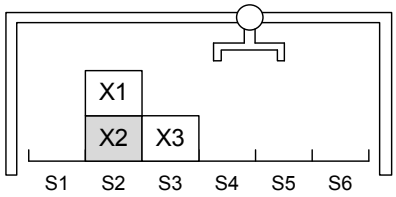


Figure 2.4: Example of problem instance in Container Stacking domain.

Since the Container Stacking Problem can be formalized with these two files, we can use a general domain independent planner to solve our problemas as *Metric FF* [29]. The plan, which is returned by the planner, is a totally ordered sequence of actions or movements which must be carried out by the crane to achieve the objective. Figure 2.5 shows an example of the obtained plan for a given problem. The performance of this general planner will be analyzed in Section 2.9, which will be compared with the domain-oriented planner that will be presented in next sections.

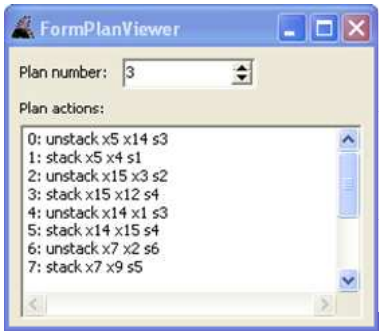


Figure 2.5: The obtained plan solution to be carried out by the transfer crane.

2.4 A Domain-Dependent Heuristically Guided Planner

Metric FF planner might obtain plans, but it is very inefficient. Therefore, we propose a domain-dependent planner in order to provide more efficiency. It means, at least reducing the number of crane operations required to achieve a desirable layout.

The proposed planner is built on the basis of a local search domain-independent planner called *Simplanner* [65]. This planner has several interesting properties for the container stacking problem:

- It is an anytime planning algorithm. This means that the planner can found a first, probably suboptimal, solution quite rapidly and this solution is being improved while time is available.
- It is complete, so it will always find a solution if exists.
- It is optimal, so that it guarantees finding the optimal plan if there is time enough for computation.

It follows an enforced hill-climbing [28] approach with some modifications:

- It applies a best-first search strategy to escape from plateaux. This search is guided by a combination of two heuristic functions and it allows the planner to escape from a local minima very efficiently.
- If a plateau exit node is found within a search limit imposed, the hill-climbing search is resumed from the exit node. Otherwise, a new local search iteration is started from the best open node.

The initial approach, based on *Simplanner*, was firstly used to solve individual subproblems (yard-bays) [58, 59]. To improve the solutions obtained by *Simplanner* we have further developed a domain-dependent heuristic to guide the search in order to accelerate and guide the search toward a optimal or sub-optimal solutions.

This heuristic (called h_1) was developed to efficiently solve one yard-bay [63]. h_1 computes an estimator of the number of container movements that must be carried out to reach a goal state (see Algorithm 1). The essential part of this algorithm is to count the number of containers located on the selected ones, but also keeps track of the containers that are held by the crane distinguishing between whether they are selected containers or not. When the crane is holding a selected container, the value h has a smaller increase since, although this state is not a solution, this container will be at the top of some row in the next movement.

2.5 Optimization criteria for one-bay yards

Despite we are able to obtain good solutions (layouts) from *Simplanner* enhanced with h_1, we also need more realistic solutions for instance taking into account safety standards.

Algorithm 1: Pseudo-code of the domain-dependent heuristic h_1

```
Data: b: state of the yard-bay;
Result: h: heuristic value of b;
h = 0;
Container hold by the crane if ∃x−container/Holding(x) ∈ b then
    if GoalContainer(x) then
        h = 0.1;
    else
        h = 0.5;
    end
end
//Increasing the Δh value for r ← 1 to numRows(b) do
    Δh = 0;
    for x−container / At(x, r) ∧ GoalContainer(x) ∈ b do
        if ∄y−container/GoalContainer(y) ∧ On(y, x) ∈ b then
            Δh = max(Δh, NumContainersOn(x));
        end
    end
    h+ = Δh;
end
```

From this heuristic h_1, we have developed some optimization criteria each of them achieving one of the requirements we could face at Container Terminals. These criteria are centered in the next issues:

1. Reducing distance of the goal containers to the cargo side (OC_{1d}) [64].

2. Increasing the range of the move actions set for the cranes allowing to move a container to 5th tier (OC_{1t}) [60].

3. Applying different ways of balancing within the same bay in order to avoid *sinks* (OC_{1b}) [54].

These criteria have been easily incorporated in our planner by defining an heuristic function as a linear combination of two functions:

$$h(s) = \alpha \cdot h_1(s) + \beta \cdot h_2(s) \tag{2.1}$$

where h_2 is a combination of above three criteria:

$$h_2(s) = OC_{1d} + OC_{1t} + OC_{1b} \tag{2.2}$$

Note that although we want to guarantee balancing with this last optimization criterion, unbalanced states (states with *sinks*) are allowed during this process of remarshalling in order to get better solutions according to the number of reshuffles.

2.5.1 OC_{1d}: Allocating goal containers close to cargo side

Given an initial state, several different layouts can be usually achieved by making the same number of reshuffles. However, some of them can be more interesting than the others according to some important questions. In this case, since the transfer crane is located at the right side of the yard-bay, we want to obtain a layout where it is minimized the distance of the goal containers to this side of the yard-bay. Achieving this we can spend considerably less time during the truck loading operations.

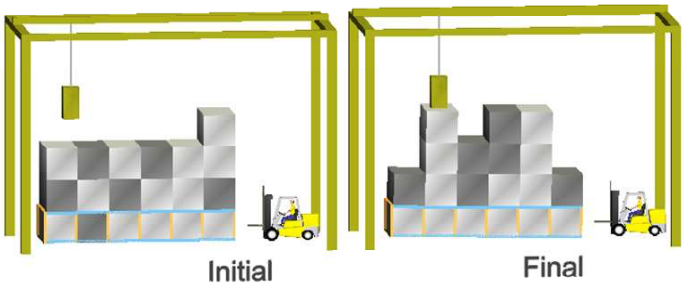


Figure 2.6: Obtained plan with the initial domain-dependent heuristic.

Following the heuristic function presented in Equation 2.1:

- $h_1(s)$ is the main heuristic function, which estimates the number of movements required to reach the goal layout (outlined in Algorithm 1). Since this is the main optimization function, α value should be significantly higher than β.
- $h_2(s)$ is the secondary function we want to optimize. In this case, it is just OC_{1d}. This means the sum of the distances of the selected containers to the right side of the yard-bay, which can be computed as Algorithm 2 shows.

Algorithm 2: Pseudo-code to calculate the distance

```
Data: s: state to evaluate
Result: d: distance value of s
d = 0;
for r ← 1 to numRows(s) do
    for x−container / At(x, r) ∈ s ∧ GoalContainer(x) do
        d = d + (numRows(s) − r);
    end
end
```

The benefits of using this combined heuristic function can be observed in Figure 2.6 and Figure 2.7. In the first one, we only want to minimize the number of reshuffles, i.e. $h(s) = h_1(s)$. In the second one, we also want to minimize the distance of the selected containers to the forklift truck, so we have set $h(s) = 9 * h_1(s) + h_2(s)$. As a result, none of the selected containers (the red ones) are placed in the most left rows, reducing the required time to load the truck.

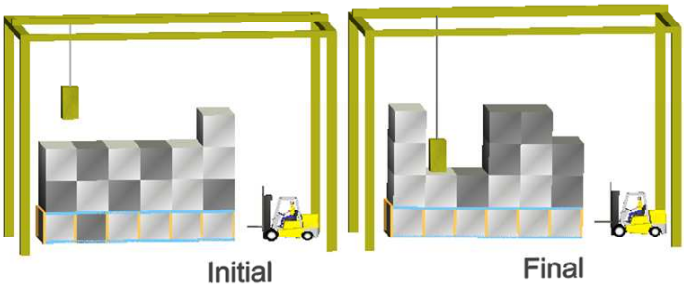


Figure 2.7: Obtained plan with the distance optimization function.

2.5.2 OC_{1t}: Allowing the 5th tier during the remarshalling process

In this optimization criterion as well as the next ones, we will include the new given heuristic value with the same factor as the initial one. One of the decisions that must be done in Container Terminals is about what type of cranes must be bought depending on how many tiers cranes work. This topic has been considered in [59]. But, another approach is to reach the fifth tier only during the remarshalling process. Thereby, there would be 4 tiers at the beginning and at the end of the process keeping the first requirements.

Following this concept, we will use instances of problems $< n, 4 >$ (where n is the number of containers) with a domain whose move actions allow 5 tiers at the stacks. This function is showed in Algorithm 3 and it follows the same steps than the original, but increasing the value of h when the height of one of the stacks is higher than 4. Thereby, we assure that the final layout will always have 4 tiers.

2.5.3 OC_{1b}: Balancing one yard-bay

In this section, we present an extension for the heuristic h_1 (Algorithm 1) to include the balancing of the stacks within one yard-bay as a requirement. It is considered that there is a *sink* when the height difference between two adjacent stacks in the same yard-bay is greater than a maximum number of containers, in our case two containers.

Considering the time when the goal containers are removed from the yard, we can distinguish three ways to get balanced one yard-bay presented in the next subsections. The last mode is the consequence of applying the first two ones.

1. **Balanced before loading operation.** In this case, we consider that the *layout must be balanced before the goal containers are removed from that yard-bay*. This function is showed in Algorithm 4. It compares the height of each row of the yard-bay with the next one, and if the difference is higher than 2, the value heuristic h is increased. As it appears in Figure 2.8, this criterion avoids the *sinks* in the final layout while all the containers are still in the yard-bay.

Algorithm 3: Pseudo-code of the domain-dependent heuristic function to allow 5 tiers

```
Data: s: state to evaluate
Result: h: heuristic value of s
h = 0;
if ∃x−container/Holding(x) ∈ s then
    if GoalContainer(x) then
        h = 0.1;
    else
        h = 0.5;
    end
end
for r ← 1 to numRows(s) do
    Δh = 0;
    if Height[r, s] > 4 then
        if x−container / Clear(x, r) ∈ s ∧ GoalContainer(x) then
            Δh = 0.5;
        else
            Δh = 1;
        end
    end
    for x−container / At(x, r) ∈ s ∧ GoalContainer(x) do
        if ∄y−container/GoalContainer(y) ∧ On(y, x) ∈ s then
            Δh = max(Δh, NumContainersOn(x));
        end
    end
    h+ = Δh;
end
```

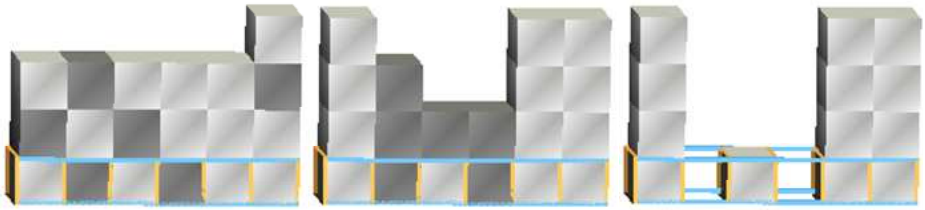
However, when these containers are removed, it might cause that the new layout is unbalanced as it happens in Figure 2.8(c).

Algorithm 4: Pseudo-code to balance before the goal containers are removed

```
Data: s: state to evaluate; h: Initial heuristic;
Result: h: heuristic value of s;
for r ← 1 to numRows(s) − 1 do
    Δh = Abs(Height[r, s] − Height[r + 1, s]);
    if Δh > 2 then
        h = h + Δh − 2;
    end
end
```

2. **Balanced after loading operation.** In contrast to the method seen above, we can consider that the *layout must remain balanced after the goal containers are removed from the yard-bay*. Figure 2.9 shows the layouts we get after execute the plan returned by our planner.

   Algorithm 6 shows this function. It uses the Function *HeightsWithoutGoals* (Algorithm 5) in order to calculate for the yard-bay b the height for each stack where the first no-goal container is. These values are employed to get the difference of height between two adjacent stacks once the goal containers have been removed from the yard. Heights of each row are stored as



(a) Initial Layout (b) With goal containers (c) Without goal containers

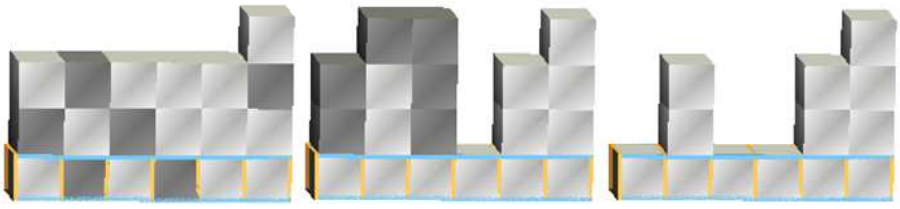
Figure 2.8: Effects of using function seen in Algorithm 4.

Algorithm 5: Function `HeightsWithoutGoals` to calculate heights of each row without taking into account the goal containers at the top

```
Data: b: state of the yard-bay;
Result: MinHeight, heights calculated;
for r ← 1 to numRows(b) do
    MinHeight[r, b] = Height[r, b];
    //Decrease till the first no goal-container
    while MinHeight[r, b] > 0 ∧ GoalContainer(MinHeight[r, b], r) ∈ b do
        MinHeight[r, b] − −;
    end
end
```

soon as the planner gets the final solution plan for one yard-bay. After we obtain these values, we increase the heuristic value h according to wether or not there are goal containers on the floor. Then, we use the values given by *HeightsWithoutGoals* to calculate the difference between two adjacent stacks, when this difference is higher than 2, we consider that there is a *sink*, so h is increased again.

However, this process might also cause some unbalanced layouts (Figure 2.9(b)). But in this case, non-desirable layouts will appear while the goal containers are in the yard-bay. Once they have been removed from it, these layouts will be balanced ones (Figure 2.9(c)).



(a) Initial Layout (b) With goal containers (c) Without goal containers

Figure 2.9: Effects of using function seen in Algorithm 6.

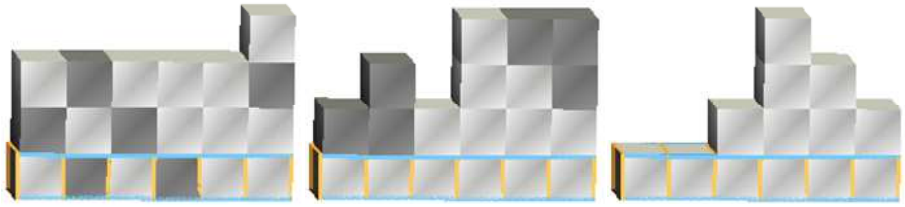
3. **Balanced before and after loading operation.** Finally, we present an optimization criterion which obtains a *layout where is balanced both before and after the goal containers are removed from this yard-bay*. With this function

Algorithm 6: Pseudo-code to balance after the goal containers are removed

```
Data: s: state to evaluate; h: Initial heuristic;
Result: h: heuristic value of s;
HeightsWithoutGoals(s);
Δh = 0;
//Not allow containers on the floor
for r ← 1 to numRows(s) do
    if ∃x−container/On(x, r) ∧ GoalContainer(x) then
        Δh = Δh + NumContainersOn(x);
    end
end
h = h + Δh;
for r ← 1 to numRows(s) − 1 do
    Δh = Abs(MinHeight[r, s] − MinHeight[r + 1, s]);
    if Δh > 2 then
        h = h + Δh − 2;
    end
end
```

we want to solve the problems seen in the last subsections as we can see it in Figure 2.10.

This function (Algorithm 7) is a mixture of the last two ones. First, we increase h when there are goal containers on the floor. When this is achieved, we increase h when the difference between the heights values obtained by the function *HeightsWithoutGoals* (Algorithm 5) are higher than 2 for two contiguos rows. And finally, if h value is low enough (in our case lower than 1), we increase h again if the difference between the actual heights of two contiguos rows is higher than 2.



(a) Initial Layout (b) With goal containers (c) Without goal containers

Figure 2.10: Effects of using function seen in Algorithm 7.

2.6 Optimization criteria for one block

This initial heuristic (h_1) was unable to solve a complete yard or block (in our case, one block consists of 20 yard-bays) due to the fact that they only solve individual yard-bays. In this section, we have also developed two optimization criteria that include new constraints that involve several yard-bays [50]. These constraints are:

- Balancing contiguous yard-bays: rows of adjacent yard-bays must be bal-

Algorithm 7: Pseudo-code to balance the yard-bay before and after the goal containers are removed

```
Data: s: state to evaluate; h: Initial heuristic;
Result: h: heuristic value of s;
HeightsWithoutGoals(s);
Δh = 0;
//Not allow containers on the floor
for r ← 1 to numRows(s) do
    if ∃x−container / On(x, r) ∧ GoalContainer(x) then
        Δh = Δh + NumContainersOn(x);
    end
end
h = h + Δh;
if h == 0 then
    Δh = 0;
    //Balancing with containers which are not objective
    for r ← 1 to numRows(s) − 1 do
        Δh = Abs(MinHeight[r, s] − MinHeight[r + 1, s]);
        if Δh > 2 then
            h = h + (Δh − 2)/2;
        end
    end
    if h < 1 then
        //Balancing with containers which are objective
        for r ← 1 to numRows(s) − 1 do
            Δh = Abs(Height[r, s] − Height[r + 1, s]);
            if Δh > 2 then
                h = h + (Δh − 2)/2;
            end
        end
    end
end
```

anced, that is, the difference between the number of containers of row j in yard-bay i and row j in yard-bay $i-1$ must be lower than a maximum (in our case lower than 3). Figure 2.11 shows which rows must be get balanced when we consider one yard-bay and Figure 2.12 left shows an example of non-balanced yard-bays (rows in dotted points).

- Dangerous containers: two dangerous containers must maintain a minimum security distance. Figure 2.12 right shows an example of two dangerous containers that does not satisfy the security distance constraint.

These constraints interrelate the yard-bays so the problem must be solved as a complete problem. However, it is a combinatorial problem and it is not possible to find an optimal or sub-optimal solution in a reasonable time. Following the previous philosophy of solving each subproblem independently (each yard-bay separately), we can distribute the problem into subproblems and solve them sequentially taken into account related yard-bays. Thus a solution to the first yard-bay is taken into account to solve the second yard-bay. A solution to the second yard-bay is taken into account to solve the third yard-bay. Furthermore, if there exist a dangerous container in a first bay, its location is taken into account to solve

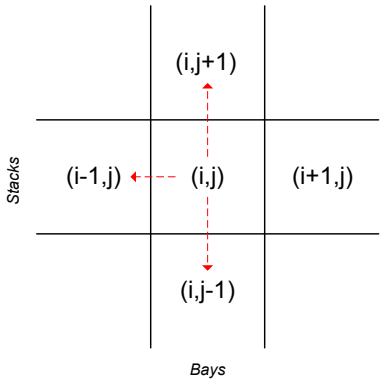


Figure 2.11: Balancing scheme.

a dangerous container located in the third yard-bay (if it exists); and so on. Taken into account this distributed and synchronous model, we present two different optimization criteria to manage these type of constraints.

These two criteria are added to the heuristic function presented in Equation 2.1 as h_3 (Equation 2.3); and Equation 2.4 shows the exact combination of them. This makes possible to follow a criterion with major priority than the other one.

$$h = \alpha \cdot h_1 + \beta \cdot h_2 + \gamma \cdot h_3 \tag{2.3}$$

$$h_3 = \delta_1 \cdot OC_{nB} + \delta_2 \cdot OC_{nD} \tag{2.4}$$

Following the proposed model and depending on the sequence of yard-bays to be analyzed, it is possible that no solutions exists. Moreover, as mentioned in Section 2.5, although we want to guarantee balancing and/or minimum distance between dangerous containers, during relocation of container process, we will allow the presence of non-desirable sates, e.g. with some *sinks* between two contiguous rows or bays. These intermediate states are allowed because through them we will be able to get better solutions taking into account as metric function the number of reshuffles done.

2.6.1 OC_{nB}: Balancing contiguous yard-bays

In this section we present an extension for the heuristic h_1 (Algorithm 1) to include the balancing of continuous yard-bays as a requirement. As we have pointed out before, it is considered that there is a *sink* when a difference higher than two containers exists between two adjacent rows in contiguous yard-bays. This criterion is an extension of the *balanced heuristic* presented in Algorithm 7, which avoids *sinks* in the same yard-bay (horizontal balance) both before and after the outbound containers have been removed from the yard. However, in this case a *sink* represents a constraint between two subproblems. Thus, we also consider that there is

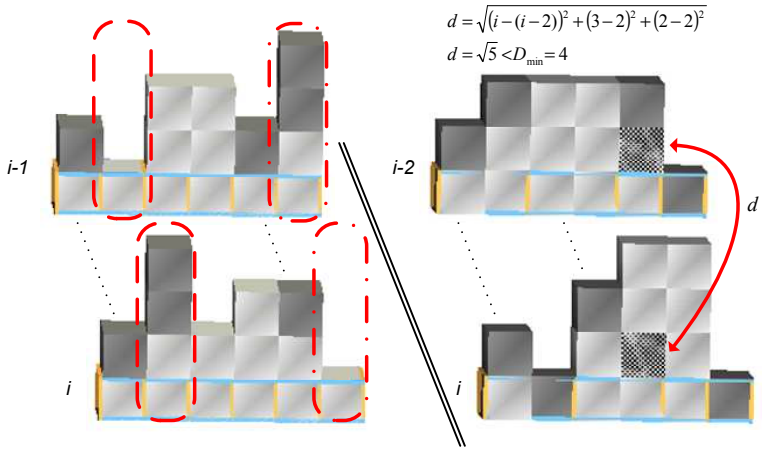


Figure 2.12: (Left) Non-balanced yard-bays. (Right) Proximity of two dangerous containers.

a *sink* when a difference of two exits between the same row r in two contiguous yard-bays (vertical balance).

This process is showed in Algorithm 8. It also uses the Function *HeightsWithoutGoals* (Algorithm 5) in order to calculate for the yard-bay b the height for each stack where the first no-goal container is. Heights of each row are stored as soon as the planner gets the final solution plan for one yard-bay.

First, we apply the criterion seen in Algorithm 7 on the yard-bay b. Through `heights'` calculated by Algorithm 5 and the real heights of the actual yard-bay we obtain the differences between the row r and $r-1$ to calculate the value of h. When this value is zero (the yard-bay b is horizontally balanced), then we introduce our function to balance it with respect to the last yard-bay b_l. To do so, we must also calculate the `heights'` through the Algorithm 5 over b_l and use the real heights of it in order to obtain the differences between the row r situated in b and b_l. When these differences are higher than 2 we increase h proportionally. After that process, b will be balanced horizontally with respect to their rows, and vertically with respect to the last yard-bay. Repeating this process for each yard-bay in the block, this will be completely balanced.

2.6.2 OC_{nD}: Dangerous containers

Within a block, there are different types of containers depending on the goods they transport, being some of them dangerous. If they do not satisfy certain restrictions, it may become a hazard situation for the yard since e.g. if one of them explodes and they are not far enough between them, it will set off a chain of explosions.

With this added objective, the next optimization criterion (Algorithm 9) ensures a minimum distance (D_{min}) between every two dangerous containers (C_d) in the yard. D_{min} is set as one parameter for the planner and the distance is calculated

Algorithm 8: Pseudo-code to balance two adjacent *yard-bays*

```
Data: b: state of the actual yard-bay; h: Initial heuristic; b_l: last yard-bay;
Result: h: heuristic value of b
//Getting the balance horizontally HeightsWithoutGoals(b);
h+ = BalBeforeAfter(b);
//This heuristic will be executed after a partial solution
if h == 0 ∧ b ≠ 1 then
    Δh = 0;
    HeightsWithoutGoals(b_l);
    //Balancing with containers which are not objective
    for r ← 1 to numRows(b) do
        Δh = Abs(MinHeight[r, b_l] − MinHeight[r, b]);
        if Δh > 2 then
            h = h + (Δh − 2)/2;
        end
    end
    if h < 1 then
        //Balancing with containers which are objective
        for r ← 1 to numRows(b) do
            Δh = Abs(Height[r, b_l] − Height[r, b]);
            if Δh > 2 then
                h = h + (Δh − 2)/2;
            end
        end
    end
end
```

as the Euclidean distance, considering each container located in a 3-dimensional space (X,Y,Z) where X is the number of yard-bays, Y is the number of rows and Z is the tier.

Generally, in container terminals, at most, there is only one dangerous container in two contiguous yard-bays, so that we take into account this assumption in the development of this function.

This function increases h value when a dangerous container C_{d1} exists in a yard-bay b and the distance constraints between dangerous containers are not hold. Thereby, for each dangerous container C_{d2} allocated in the previous D_{min} yard-bays is calculated by Euclidean distance to C_{d1}. If this distance is lower than D_{min}, for any dangerous container C_{d2}, then h value is increased with the number of containers n on C_{d1} because it indicates that removing those n containers is necessary to reallocate the container C_{d1}.

2.7 An Ordered Yard-Based Planner

In this section we improve the above planner to efficiently manage a full container yard [61]. A block of containers is decomposed of several yard-bays, so that the problem is distributed into a set of subproblems. Thus, each yard-bay generates a subproblem. The order of execution of yard-bays can be sequential or it can be ordered by tightness. Figure 2.13 shows two different ways to manage a complete yard. Figure 2.13 (upper) shows a sequential order of execution of yard bays (Plan-

Algorithm 9: Pseudo-code to avoid locating two dangerous containers closer to a distance D_{min}

```
Data: B: whole block; b: state of the actual yard-bay; h: Initial heuristic; D_min: Minimum
      distance; NC: Number of containers;
Result: h: heuristic value of b;
nBay = NumBay(b);
if nBay > 1 ∧ h < NC ∧ ∃C_d1 ∈ b then
    Δh = 0;
    L_1 = Location(C_d1);
    foreach b_l ∈ Y/NumBay(b_l) ∈ {max(nBay − D_min + 1, 1), nBay − 1} do
        if ∃C_d2 ∈ b_l then
            L_2 = Location(C_d2);
            dist = EuclideanDistance(L_1, L_2);
            if dist < D_min then
                Δh = Δh + NumContainersOn(C_d1);
            end
        end
    end
    h+ = Δh;
end
```

Algorithm 10: Sinks within a whole block

```
Data: B: whole block;
Result: nSinks: number of Sinks;
nSinks = 0;
for b ← 1 to numYards(B) do
    for r ← 1 to numRows(b) − 1 do
        Δh = Abs(Height[r, b] − Height[r + 1, b]);
        if Δh > 2 then
            nSinks + +;
        end
    end
    if NumBay(b) > 1 then
        for r ← 1 to numRows(b) do
            Δh = Abs(Height[r, b] − Height[r, b − 1]);
            if Δh > 2 then
                nSinks + +;
            end
        end
    end
end
```

ner H1 Sequential). Following the optimization criteria OC_{nB} (Section 2.6.1). Thus, the planning of a yard-bay is carried out taken into account the solution obtained by the previous yard-bay (see Figure 2.14). Thus a first plan is obtained for the first yard-bay. Then a new plan is carried out for the second yard-bay taken into account the balance constraints generated with the solution obtained for the first yard-bay, and so on.

Figure 2.13 (lower) shows a different order of execution of yard-bays (Planner H1 Ordered). The tightest yard-bays (yard-bays with more export containers) are analyzed first. Thus, the number of reshuffles is minimized due to the fact that the tightest yard-bays are solved without the need of satisfying the balance constraints with the contiguous ones, meanwhile their neighbour yard-bays must be

Algorithm 11: Unfeasible relationships between two dangerous containers within a whole block

Data: B: whole *block*;
Result: $nDang$: number of Sinks;
$nDang = 0$;
for $b \leftarrow 1$ **to** `numYards`(B) **do**
  $nBay =$ `NumBay`(b);
  if $nBay > 1 \wedge \exists C_{d1} \in b$ **then**
    $L_1 =$ `Location`(C_{d1});
    foreach $b_l \in Y$ / `NumBay`$(b_l) \in \{\max(nBay - D_{min} + 1, 1), nBay - 1\}$ **do**
      if $\exists C_{d2} \in b_l$ **then**
        $L_2 =$ `Location`(C_{d2});
        $dist =$ `EuclideanDistance`(L_1, L_2);
        if $dist < D_{min}$ **then**
          $nDang++$;
        end
      end
    end
  end
end

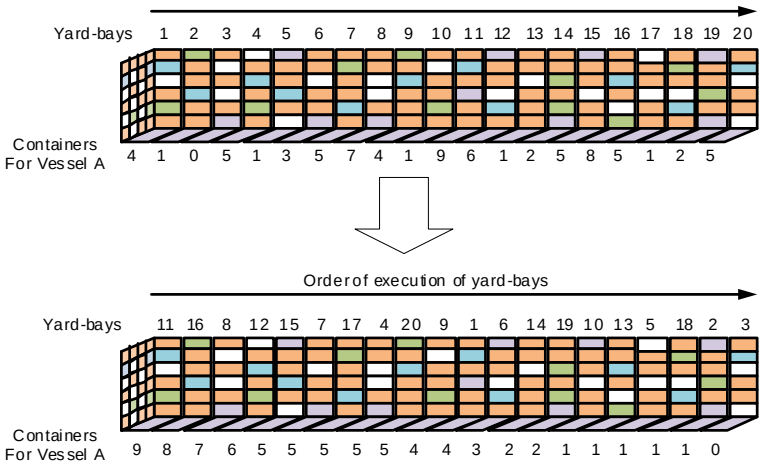


Figure 2.13: Order of execution of yard-bays.

committed to these tasks. Thus, following the example of Figure 2.13, the yard-bay 11 is executed first without balance constraints because it has 9 goal containers meanwhile yard-bays 10 and 12 are executed later taken into account the balance constraints generated by the solution of yard-bay 11. In the evaluation section we will compare the behavior of two alternatives. In general the order of execution of yard-bays is directly related with the efficiency of our planning tool. The ordering of yard-bays by tightness improves the efficiency our planning tool.

Furthermore, containers of different yard-bays must satisfy other constraints among them such as dangerous containers that must maintain a minimum security (Euclidian) distance among them (Planner H1DC). In order to insert our planner in the integrated system, we have improved our version to minimize the number of

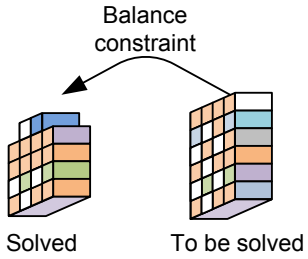


Figure 2.14: The balance constraints for continuous yard-bays.

reshuffles for a set of out containers to be loaded in different vessels. Initially our planner was developed to minimize the number of reshuffles for a vessel (vessel A in Figure 2.13). However, the order of the rest of containers in the yard-bay did not matter. Our new planner takes into account these features and it is able to organize the bay in order to adapt to the berth schedule. Thus, the reshuffles needed to allocate the out containers for a vessel are carried out taken into account the out containers for the following vessel to berth.

2.8 Analytic Formula to Estimate the number of Reshuffles

As we have pointed out, solving the CStackP is a NP-complete combinatorial optimization problem. Once the BAP returns a possible schedule of vessels to berth in the port, the study of yard reshuffles must be carried out for each vessel. This is a very hard task so that a general formula that estimates the number of reshuffles for each vessel remains necessary. Once the best solution is achieved, the planning tool is executed to obtain the optimal plan for the yard reshuffles. To this end, we have identified the main parameters that affect the number of reshuffles in a yard-bay:

1. Number of total slots in a yard-bay (P_1). For instance, a yard-bay with tier 5, the number of slots is 30.
2. Number of containers in the yard-bay (P_2). It is well-known that $P_2 \leq P_1$.
3. Number of goal (out) containers (P_3). It is well-known that $P3 \leq P2$
4. Number of containers on top of goal containers (P_4). A lower bound for the minimal number of reshuffles is P_4.

These parameters influence in the number of reshuffles needed for each yard-bay. The estimator function R is mainly depended on P_4 and it is bounded by:

$$R = P_4 + \alpha : \qquad \alpha \in [0, \infty) \tag{2.5}$$

where $\alpha = 0$ means that the problem is underconstrained meanwhile $\alpha \to \infty$ means that the problem is unsolvable.

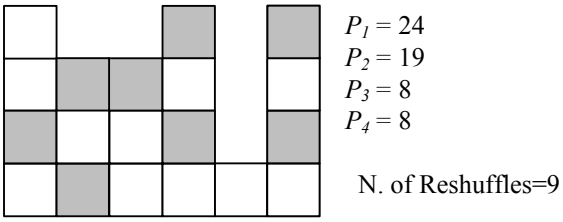


Figure 2.15: Example and values of parameters.

Figure 2.15 shows an example of yard-bay with the value of each identified parameter. The rest of parameters also play an important role but the relationship among them is not clear. The simulation of one hundred yard-bays with different tiers and number of containers and objectives return a strong relationship among the parameters P_1, P_2 and P_3. If $(P_1 - P_2)/P_3$ is lower than 1.25, one more reshuffle is needed ($P_4 + 1$). In the same way if $(P_1 - P_2)/P_3$ is lower than 1, one more reshuffle is also needed $(P_4 + 2)$; and so on. Finally if $(P_1 - P_2) \rightarrow 0$ the problem is unsolvable. Thus, we estimate the number of reshuffles by the following formula:

$$R = P_4 + \left\lfloor \frac{2}{\frac{P_1 - P_2}{P_3}} \right\rfloor = P_4 + \left\lfloor \frac{2 \cdot P_3}{P_1 - P_2} \right\rfloor \tag{2.6}$$

This estimator R is accurate enough for us to do not need to execute our planner for each berthing plan. In the integrated system presented above, we can select to run our planner or to use the estimator. In any case, once the best solution is found for the integrated problem, our planner must solve the best solution in order to determine the specific plan that the cranes must be carried out to allocate the containers in the appropriate places.

2.9 Evaluation

In this section, we evaluate the behavior of the heuristic with the set of optimization criteria presented. The experiments were performed on random instances. A random instance of a yard-bay is characterized by the tuple $< n, s >$, where n is the number of containers in a yard-bay and s is the number of selected containers in the yard-bay. Each instance is a random configuration of all containers distributed along six stacks with 4 tiers. They were solved on a personal computer equipped with a Core 2 Quad Q9950 2.84Ghz with 3.25Gb RAM.

First, we present a comparison between our basic domain dependent heuristic h_1 against a domain independent one (*Metric FF*). Thus, Table 2.1 presents the average running time (in milliseconds) to achieve a first solution as well as quality of the best solution found (number of reshuffles) in 10 seconds for our domain-dependent planner. This table also shows the average running time (in millisec-

onds) and the quality of the solution for *Metric FF*. Both planners have been tested in problems $< n, 4 >$ evaluating 100 test cases for each one. Thus, we fixed the number of selected containers to 4 and we increased the number of containers n from 15 to 21.

It can be observed that our new domain-dependent heuristic is able to find a solution in a few milliseconds, meanwhile the domain-independent planner (*Metric FF*) needs much more time to find a solution. This solution also needs more moves to get a goal state. Furthermore, due to the fact that our tool is an anytime planner, we evaluate the best solution found in a given time (10 seconds).

Table 2.1: Average number of reshuffles and running time of *Metric FF* and h_1 in problems $< n, 4 >$.

Instance	***Metric FF***		**Heuristic (h_1)**	
	Running time	Solution	Running time first solution	Best Solution in 10 secs
$< 13, 4 >$	22	3.07	2	3.07
$< 15, 4 >$	3102	4.04	6	3.65
$< 17, 4 >$	4669	5.35	12	4.35
$< 19, 4 >$	6504	6.06	24	4.72
$< 20, 4 >$	22622	7.01	36	5.22
$< 21, 4 >$	13981	6.82	66	5.08

Now we show the effects of using each one of the criteria described in Section 2.5 separately. In Table 2.2, we present the average sum of distances between the selected containers and the right side of the layout in both our domain-independent heuristic and our domain-dependent heuristic with distance optimization for problems $< n, 4 >$. As mentioned above, we fixed the number of selected containers to 4 and we increased the number of containers n from 13 to 21. It can be observed that distance optimization function helps us for finding solutions that place the selected containers closer to the cargo side of the yard-bay.

Applying the criterion or function showed in Algorithm 3 we obtain the results appeared in Table 2.3. These results are the comparison between the number of solved problems over 100 problems $< n, 4 >$ using or not that criterion in just one second. Through this table we can conclude that:

- The higher number of containers, the lower problems are solved. This is because as we increase the number of containers there are less positions or gaps where containers could be remarshalled.

- Allowing movements to the 5^{th} helps us to solve more problems. It is remarkable with instances $< 23, 4 >$ with H_1 only three problems could be solved, however OC_{1t} solves 84 over 100 problems.

Table 2.2: Average distance obtained by considering distance or not in our domain-dependent heuristic $< n, 4 >$ with 4 tiers.

Instance	***Metric FF***		$\boldsymbol{OC_{1d}}$	
	Distance	*Reshuffles*	*Distance*	*Reshuffles*
$< 13, 4 >$	11.28	3.07	10.91	3.07
$< 15, 4 >$	10.60	4.04	9.21	3.65
$< 17, 4 >$	10.58	5.35	8.87	4.46
$< 19, 4 >$	12.28	6.06	8.32	4.86
$< 20, 4 >$	12.71	7.01	7.75	5.56
$< 21, 4 >$	12.20	6.82	8.36	5.34

Table 2.3: Number of solved problems $< n, 4 >$ with 4 and 5 tiers during the process.

Instance	**4 tiers $\boldsymbol{h_1}$**	**5 tiers $\boldsymbol{OC_{1t}}$**
$< 19, 4 >$	100	100
$< 20, 4 >$	100	100
$< 21, 4 >$	95	99
$< 23, 4 >$	3	84

Last criterion for solving problems where we only take into account one yard-bay is showed in Section 2.5.3. Since Algorithm 7 presents the best results after the whole process of remarshalling, we do the comparison in Table 2.4 among the solutions given by *Metric FF* planner, the initial one h_1 and OC_{1b} (*Both*) in 50 test cases. The last two ones look for solutions during 1 second in instances of $< 15, 4 >$ and 4 seconds in instances of $< 17, 4 >$. These times are used in order to achieve a solution for all the instances.

Table 2.4: Average number of movements, sinks and time for the first solution in problems $< 15, 4 >$ (1) and $< 17, 4 >$ (2) using or not balanced heuristics.

	Metric FF		$\boldsymbol{h_1}$		$\boldsymbol{OC_{1b}}$	
	(1)	(2)	(1)	(2)	(1)	(2)
Reshuffles	3.72	4.24	3.42	3.68	5.36	5.30
Sinks	0.62	0.50	0.92	0.66	0	0
Time First Sol.	2621	2961	6	10	16	22

Sinks are calculated by Algorithm 10. As we mentioned above, we consider that there is a sink where the difference in tiers between two adjacent rows is higher than 2. Thereby, in this algorithm we are counting sinks produced between two contiguos stacks at the same yard-bay as well as between two rows in one yard-bay

and the previous one. This process takes into account the goal containers in final yard-bays.

From here, we realize an evaluation for the criteria presented in Section 2.6. Table 2.5 shows the performance of the criteria for solving the whole block of yard-bays. These experiments were performed in blocks of 20 yard-bays and each one of them are instances $< 15, 4 >$. This evaluation was carried out in a yard with 3 blocks of 20 yard-bays. The number of unfeasible relationships between dangerous containers is calculated by means of Algorithm 11. Basically, we look for those pairs of dangerous containers whose distance between them is shorter than minimum distance (D_{min}).

The results showed in Table 2.5 represent the average number of reshuffles, the average number of sinks generated along the block and the average number of unsatisfied dangerous containers. Results given by these optimization criteria are the average of the best solutions found in 10 seconds. It can be observed that h_1 still outperforms *Metric FF* in the average number of reshuffles. However, due to the fact that they do not take into account the balancing constraints, *Metric FF* generated an average of 24.33 sinks in the block of yard-bay and h_1 generated and average of 32.67 sinks. And it occurs the same for the average number of unfeasible constraints for dangerous containers, *Metric FF* gives us 15.33 and h_1 obtains 7.67.

Taking into account that OC_N is a junction of OC_{nB} and OC_{nD}, both OC_{nB} and OC_{nD} solved their problems. That is, OC_{nB} obtained its solutions with no sinks and OC_{nD} obtained its solutions by satisfying all dangerous constraints. Furthermore, OC_N was able to solve its problems by satisfying both types of constraints. However we could state that balancing problem is harder than the problem related to dangerous containers because OC_{nB} needs more reshuffles to obtain a solution plan than OC_{nD}. Moreover, we observe with OC_{nB}, OC_{nD} and OC_N ensure the established requirements however the average reshuffles is increased with respect to h_1.

Table 2.5: Average results with blocks of 20 yard-bays each one being a $< 15, 4 >$ problem.

	Metric FF	h_1	OC_{nB}	OC_{nD}	OC_N
Reshuffles	3.98	3.60	5.68	4.30	6.53
Sinks	24.33	32.67	0	33.33	0
Non-Safe Dangerous	15.33	7.67	8.00	0	0

In Table 2.6, we show the performance of our planner H1 Sequential and H1 Ordered (Section 2.7). These experiments were performed on blocks of containers composed by 10 yard-bays in the form $< 17, s >$. Thereby, each yard-bay has a different number of goal containers. Furthermore, we have established a timeout of 35 seconds to solve each yard-bay.

The first column in Table 2.6 corresponds to each instance solved by each plan-

Table 2.6: Comparison of H1 Sequential and H1 Ordered.

Instance	Order (Yard-Bay/N. Goal containers)										Number
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Reshuffles**
1-O	6	1	2	8	3	4	10	9	5	7	91
	9	7	6	6	5	5	5	4	2	2	
1-S	1	2	3	4	5	6	7	8	9	10	*Time Out*
	7	6	5	5	2	9	2	6	4	5	
2-O	2	5	6	1	4	3	8	9	10	7	64
	7	7	7	6	6	5	4	4	3	2	
2-S	1	2	3	4	5	6	7	8	9	10	*Time Out*
	6	7	5	6	7	7	2	4	4	3	
3-O	3	2	7	8	10	4	6	1	9	5	55
	8	6	6	6	6	5	5	4	4	1	
3-S	1	2	3	4	5	6	7	8	9	10	99
	4	6	8	5	1	5	6	6	4	6	
4-O	7	9	2	10	3	4	6	1	8	5	63
	10	7	6	6	5	5	4	3	3	2	
4-S	1	2	3	4	5	6	7	8	9	10	63
	3	6	5	5	2	4	10	3	7	6	
5-O	7	1	10	2	8	9	4	5	6	3	78
	9	7	7	6	6	6	5	5	4	2	
5-S	1	2	3	4	5	6	7	8	9	10	*Time Out*
	7	6	2	5	5	4	9	6	6	7	

ner. Thereby, row 1 corresponds to instance 1 of planner H1 Ordered (1-O), row 2 corresponds to instance 1 of planner H1 Sequential (1-S), and so on. The following 10 columns have two rows for each instance. They show for each instance the order in which the yard-bays are executed (upper row) and the number of goal containers that each one of them has (lower row). As example, for the instance 1-O, the sixth yard-bay is the first one in being executed and it has 9 goal containers. Finally, the last column presents the number of reshuffles needed to solve the instance or *Time Out* if a solution is not found.

As it can be observed, there are instances which only can be solved through the H1 Ordered, e.g. instances 1, 2 and 5. Moreover, other instances (instance 3), through the H1 Ordered planner, give a more efficient plan than the sequential one. However, there are also other examples in which both planners return the same plan (instance 4).

Thus, we can conclude that H1 Ordered can be considered a better planner than H1 Sequential to solve the complete block of yard-bays.

The actual average number of reshuffles and the average value of our estimator R are presented in Table 2.7. In each row, we present the average number of reshuffles from a set of 100 random instances. In all cases, we considered yard-bays with tier 4, so that the number of possible containers to be allocated to each yard-bay is set to 24 ($P_1 = 24$). The rest of parameters were increased: P_2 from 15 to 19 and P_3 from 4 to 8. It can be observed the similitude of the average number of reshuffles and R in all cases. It can be observed that the estimator R achieved values close to the actual values in all cases. The average value of R in all instances was very similar to the average value of the actual number of reshuffles. The standard deviation of R was even lower than the actual number of reshuffles due to the fact that it is not dependent on the original allocation of out containers.

Table 2.7: Values obtained through estimator R.

	$\mathbf{P_2 = 15}$		$\mathbf{P_2 = 17}$		$\mathbf{P_2 = 19}$	
	Reshuffles	R	**Reshuffles**	R	**Reshuffles**	R
$\mathbf{P_3 = 4}$	2.8	2.6	3.7	4.6	4.6	4.8
$\mathbf{P_3 = 6}$	4.2	4.8	6.2	5.9	5.6	6
$\mathbf{P_3 = 8}$	6.2	6	8.6	8	7.2	8
Avg. values	4.4	4.5	6.2	6.2	5.8	6.3
St. deviation	2.1	1.9	3.1	2.1	2.6	1.9

2.10 Conclusions

This chapter presents a domain-dependent heuristic and a set of optimization criteria for solving the container stacking problem by means of Artificial Intelligence planning techniques. We have developed a domain-dependent planning tool for finding optimized plans to obtain an appropriate configuration of containers in a yard-bay. Thus, given a set of outgoing containers, our planner minimizes the number of necessary reshuffles of containers in order to allocate all selected containers at the top of the stacks. This proposed planner is able to satisfy both balancing constraints and dangerous container constraints, as well as reducing the distance of the goal containers to the cargo side or allowing a fifth tier during the remarshalling process.

Additional criteria have been defined for management blocks of yard-bays into consideration. However, as the problems involve a larger number of constraints, the solution becomes harder and the number of reshuffles increases. Due to the fact that a solution of a yard-bay influences on the solution of the following yard-bay, the order of solving the yard-bays will determine the minimal number of reshuffles.

This proposed planner with a domain-dependent heuristic allows us to obtain optimized and efficient solutions. This automatic planner can help to take decisions in the port operations dealing with real problems. Moreover, it can help to simulate operations to obtain conclusions about the operation of the terminal, evaluate alternative configurations, obtain performance measures, etc. Particularly, in [59] the proposed planner has been applied for obtaining an evaluation of alternative 4 or 5 tiers stacks configuration.

Chapter 3

Berth Allocation and Quay Crane Assignment Problem

3.1 Introduction to BAP and QCAP

Container terminals are open systems with three distinguishable areas (see Figure 3.1): the berth area, where vessels are berthed for service; the storage yard, where containers are stored as they temporarily wait to be exported or imported; and the terminal receipt and delivery gate area, which connects the container terminal to the hinterland. Each one of them presents different planning and scheduling problems to be optimized. For example, berth allocation, quay crane assignment, stowage planning and quay crane scheduling must be managed in the berthing area; the container stacking problem, yard crane scheduling and horizontal transport operations must be carried out in the yard area; and hinterland operations must be solved in landside area. Figure 3.2 shows the main planning and scheduling problems that must be managed in the berth area.

We will focus our attention to the Berth Allocation Problem (BAP), a well-known NP-Hard combinatorial optimization problem, which consists of assigning incoming vessels to berthing positions. Once a vessel arrives at the port, it enters in the harbor waiting time to moor at the quay. The quay is a platform protruding into the water to facilitate the loading and unloading of cargo. The locations where mooring can take place are called berths. These are equipped with giant cranes, called pier or quay cranes (QC), used to load and unload containers which are transferred to and from the yard by a fleet of vehicles. In a transshipment terminal the yard allows temporary storage before containers are transferred to another ship or to another mode (e.g., rail or road).

Managers at container terminals face two interrelated decisions: *where* and *when* the vessels should moor. First, they have to take into account physical restrictions as length or draft, but also they must take into account the priorities and other aspects to minimize both port and user costs, which are usually opposites. Nowadays, this process is usually solved manually with a first come, first served

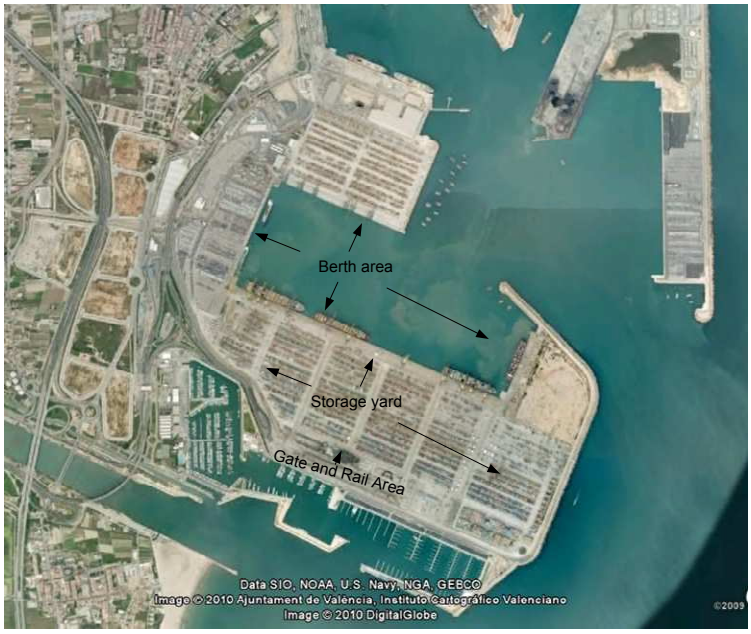


Figure 3.1: Container Terminal at Valencia

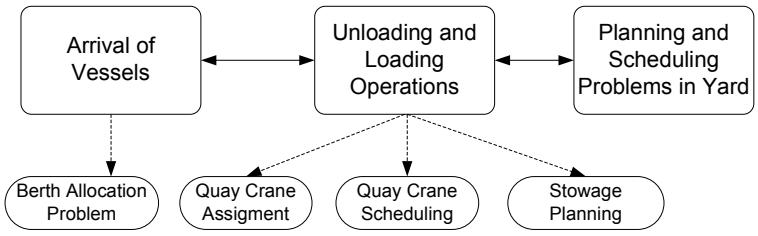


Figure 3.2: Planning and scheduling problems in Container Terminals

policy . That is, the order the vessels arrive is the same they are moored. Figure 3.3 shows an example of graphical space-time representation of a berth planning with 6 vessels. Each rectangle represents a vessel with its handling time and length.

The reminder of this chapter is organized as follows: the next section presents a literature review about the BAP and QCAP and different techniques to manage them. Section 3.3 address the developed modelas as well as their notations. Section 3.4 and Section 3.5 describe the notation used and the developed metaheuristic techniques, respectively. In section 3.6 the computational results are reported and finally, conclusions are presented.

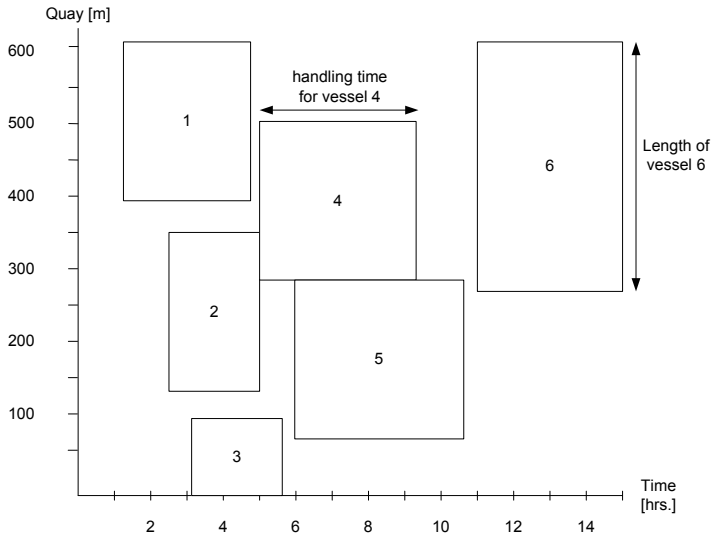


Figure 3.3: A berth planning

3.2 Literature review

In [68], the authors show a complete comparative study about different solutions for the BAP according to their efficiency in addressing key operational and tactical questions relating to vessel service. They also study the relevance and applicability of the solutions to the different strategies and contractual service arrangements between terminal operators and shipping lines.

To show similarities and differences in the existing models for berth allocation, Bierwirth and Meisel [3] developed a classification scheme according to four attributes (see Figure 3.4). The spatial attribute concerns the berth layout and water depth restrictions. The temporal attribute describes the temporal constraints for the service process of vessels. The handling time attribute determines the way vessel handling times are considered in the problem. The fourth attribute defines the performance measure to reflect different service quality criteria. The most important ones are focused on minimizing the waiting time and the handling time of a vessel. Both measures aims at providing a competitive service to vessel operators. If both objectives are pursued (i.e. wait and hand are set), the port stay time of vessels is minimized. Other measures are focused on minimizing the completion times of vessels among others. Thus, by using the above classification scheme, a certain type of BAP is described by a selection of values for each one of the attributes. For instance, let's be a problem where the quay is assumed to be a continuous line (*cont*). The arrival times restrict the earliest berthing of vessels (*dyn*) and handling times depends on the berthing position of the vessel (*pos*). The objective is to minimize the sum of the waiting times (*wait*) and handling time (*hand*). According to the scheme proposed by [3], this problem is classified by $cont|dyn|pos|\Sigma(wait+hand)$.

Value	Description
1. Spatial attribute	
disc	The quay is partitioned in discrete berths
cont	The quay is assumed to be a continuous line
hybr	The hybrid quay mixes up properties of discrete and continuous berths
draft	Vessels with a draft exceeding a minimum water depth cannot be berthed arbitrarily
2. Temporal attribute	
stat	In static problems there are no restrictions on the berthing times
dyn	In dynamic problems arrival times restrict the earliest berthing times
due	Due dates restrict the latest allowed departure times of vessels
3. Handling time attribute	
fix	The handling time of a vessel is considered fixed
pos	The handling time of a vessel depends on its berthing position
QCAP	The handling time of a vessel depends on the assignment of QCs
QCSP	The handling time of a vessel depends on a QC operation schedule
4. Performance measure	
wait	Waiting time of a vessel
hand	Handling time of a vessel
compl	Completion time of a vessel
speed	Speedup of a vessel to reach the terminal before the expected arrival time
tard	Tardiness of a vessel against the given due date
order	Deviation between the arrival order of vessels and the service order
rej	Rejection of a vessel
res	Resource utilization effected by the service of a vessel
pos	Berthing of a vessel apart from its desired berthing position
misc	Miscellaneous

Figure 3.4: A classification scheme for BAP formulation [3].

One of the early works that appeared in the literature was focused on the development of a heuristic algorithm which a First-Come-First-Served (FCFS) rule was considered [40]. However, the idea that for high port throughput, optimal vessel-to-berth assignments should be found without considering the FCFS bases was introduced by [31]. Therefore, we will use the FCFS rule in order to get an upper bound. Nevertheless, this approach may result in some vessels' dissatisfaction regarding the order of service.

In [20], multiple vessel mooring per berth is allowed assuming that vessel arrivals can be grouped into batches. They have developed a tree search procedure which provides an exact solution and this is improved by a composite heuristic.

Some metaheuristics have been developed to solve the BAP. In [11] two Tabu Search heuristics are presented to solve the discrete and continuous case, respectively to minimize the weighted sum of the service time for every ship. Both heuristics are inspired by a *Multi-Depot Vehicle Routing Problem with Time Windows* algorithm and can handle various features of real-life problems as time windows or favorite and acceptable berthing areas. Mauri et al. [44] design a column generation approach for the problem of Cordeau et al. [11] which delivers better solutions in shorter runtime than Tabu Search. In the models of Han et al. [21] and Zhou et al. [71], a Genetic Algorithm (GA) is proposed to solve this problem. In both models, the draft of vessels restricts the berth assignment decisions.

An approach based on multi-objective optimization problem using evolutionary algorithms [7] is followed to minimize the makespan of the port, total waiting time of the ships, and degree of deviation from a predetermined service priority schedule.

In [30], Imai et al. provide a solution for the integration of BAP with the Quay Crane Assignment Problem (QCAP) based on genetic algorithms. It minimizes the weighted number of vessel rejections. In this sense, an integration through two mixed integer programming formulations including a tabu search method is presented by Giallombardo et al. [17]. The latter tabu search method is an adaption of the one of [11]. However, Giallombardo et al. minimize the yard-related housekeeping costs generated by the flows of containers exchanged between vessels. These two approximations consider the discrete case of the BAP.

Considering the quay as a continuos line, a non-linear integer programming model is developed to integrate BAP and QCAP in [48]. They do not consider restrictions on the berthing times, one vessel could be scheduled earlier or later than the committed or arrival time.

In [3], the authors give a comprehensive survey of berth allocation and quay crane assignment formulations from the literature. Some authors outline approaches more or less informally while others provide precise optimization models. More than 40 formulations are presented distributed among discrete problems, continuous problems and hybrid problems. Hansen et al. [22] considered a discrete problem with a tardiness objective which accounts for departure time related costs including penalties for tardiness as well as benefits for early departures. This problem was solved by a variable neighborhood search which turns out to be superior to the GA of Nishimura et al. [46].

Taken into account the requirements of container operators of MSC (Mediterranean Shipping Company S.A), our approach also studies the integration of these two problems (BAP and QCAP) through a metaheuristic called Greedy Randomized Adaptive Search Procedures (GRASP) [13]. This metaheuristic is able to find feasible solutions within an acceptable computational time. Following the above classification scheme (see Figure 3.4), our approach is classified by $cont|dyn|QCAP|\Sigma$ *wait*. Thus, we focused on the following attributes and performance measure:

- **Spatial attribute: cont**: we assume the quay is a continuous line, so that there is no partitioning of the quay and vessel can berth at arbitrary positions within the boundaries of the quay. It must be taken into account that for a continuous layout, berth planning is more complicated than for a discrete layout at the advantage of better utilizing quay space [3].

- **Temporal attribute: dyn**: we assume dynamic problems where arrival times restrict the earliest berthing times. Thus, fixed arrival times are given for the vessels. Hence, vessels cannot berth before their expected arrival time.

- **Handling time attribute: QCAP**: we assume the handling time of a vessel depends on the assignment of QCs.
- **Performance measure: wait**: Our objective is to minimize the sum of the waiting time of all scheduled vessels to be served.

3.3 BAP-QCAP models

The objective in BAP is to obtain an optimal distribution of docks and cranes to vessels. This problem can be considered as a special kind of machine scheduling problem, with specific constrains (length and depth of vessels, ensure a correct order for vessels that exchange containers, assuring departing times, etc.) and optimization criteria (priorities, minimization of waiting and staying times of vessels, satisfaction on order of berthing, minimizing cranes moves, degree of deviation from a pre-determined service priority, etc.).

The First-Come-First-Served (FCFS) rule can be used to obtain an upper bound of the function cost in BAP [40]. Several methods have been proposed in the literature for solving BAP. Usually, these methods are mainly based on heuristic [20] or metaheuristic [11], [7] approaches.

Our approach follows an integration of the Quay Crane Assignment Problem (QCAP) and the BAP through the metaheuristic Greedy Randomized Adaptive Search Procedure (GRASP) [13] which is able to obtain optimized solutions in an efficient way.

3.3.1 BAP+*static* QCAP

In this section we present the notation of the main parameters (Figure 3.5) that will be used in the proposed metaheuristic techniques. In this first model ($\mathcal{M}_v$), known as *static QCAP* [51, 53], QCs are assigned to one vessel and they can not be moved to another vessel until the former one leaves the Container Terminal.

Let V_i be an incoming vessel. We define:

- $a(V_i)$: Arrival time of the vessel V_i at port.
- $m(V_i)$: Moored time of V_i.
- $pos(V_i)$: Berthing position where V_i will moor.
- $c(V_i)$: Number of required movements to load and unload containers of V_i.
- $q(V_i)$: Number of assigned QCs to V_i.
- $d(V_i)$: Departure time of V_i, which depends on $m(V_i)$, $c(V_i)$ and $q(V_i)$.
- $w(V_i)$: Waiting time V_i arrives at port until it moors:

$$w(V_i) = m(V_i) - a(V_i) \tag{3.1}$$

- $l(V_i)$: Length of V_i.
- $pr(V_i)$: Vessel's priority.
- QC : Available QCs in the Container Terminal.
- L : Total length of the berth in the Container Terminal.

Berthing position ($pos(V_i)$) will be determined according to the length of the vessels (and their security distances) which have previously planned. In addition, this position will be as close to the ends of the quay as possible. Thereby, avoiding one vessel could moor in the middle of the quay gives more continuous length available to moor the remaining vessels.

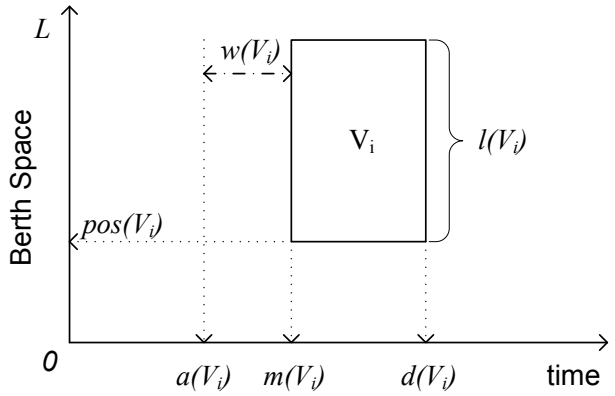


Figure 3.5: Representation of a vessel according its position and times

Basically, our objective is to allocate all vessels according to several constraints with the objective of minimizing the total weighted waiting time. To this end, let's assume a priority of each vessel according to its length and the number of movements (loading and unloading operations), in order to avoid the vessels' dissatisfaction mentioned above, as:

$$pr(V_i) = \alpha \times l(V_i) + \beta \times c(V_i) \tag{3.2}$$

where α and β are the needed factors to distribute the values of $l(V_i)$ and $c(V_i)$ in order to range the priority in $[1, 10]$. For instance, the vessels with length $0 < l(V_i) \leq 100$ the $\alpha \times l(V_i) = 1$; the vessels with the number of containers $0 < c(V_i) \leq 200$ the $\alpha \times c(V_i) = 1$; therefore, the priority value would be 2.

As we have pointed out, we deal with continuous and dynamic BAP and the time is discretized into integer units $(1, 2, \ldots, T)$. Moreover, we consider the following assumptions:

- Number of quay cranes (QC) assigned to a vessel do not vary along all the moored time. Moreover, all QC do the same number of movements by unit time (`movsQC`).

- All the information related to the waiting vessels is known in advance.
- Every vessel has a draft lower or equal than the quay.
- Mooring and unmooring are no time consuming.
- Simultaneous berthing is allowed.

Therefore, in order to allocate one vessel at berth, the following constraints must be accomplished:

- Moored time must be at least the same that its arrival time:

$$m(V_i) \geq a(V_i)$$

- There is enough contiguous space at berth to moor the vessel ($l(V_i)$).
- There is a security distance (`secLength`) between two moored ships: let's assume 5% of their lengths (the maximum of these two contiguous vessels).
- There must be at least one QC to be assigned to each vessel. The maximum number of assigned QC by vessel depends on its length. This is due to the fact that a security distance is required between two contiguous QC (`secQC`) and the maximum number of cranes that the Container Terminal allows per vessel (`maxQC`). Moreover, once a crane starts to work into a vessel, it must complete it without any pause or shift (jobs non-preemptive). Thus, the handling time of V_i is given by:

$$\frac{c(V_i)}{q(V_i) \times \texttt{movsQC}} \tag{3.3}$$

The goal of the BAP is to allocate each incoming vessel according to the existing constraints by minimizing the total weighted waiting time of vessels:

$$T_w = \sum_i w(V_i)^\gamma \times pr(V_i) \tag{3.4}$$

where $\gamma : (\gamma \geq 1)$ is an adjustment factor to prevent that lower priority vessels are systematically delayed. Note that this objective function is different to the tardiness concept in scheduling. The weighted optimization of tardiness of vessels would be:

$$T_t = \sum_i w(V_i) \times (d(V_i) - dueTime(V_i)) \tag{3.5}$$

Thus, the departure time of vessels $d(V_i)$ with respect to their due times $dueTime(Vi)$ is optimized.

3.3.2 BAP+*dynamic* QCAP

In this section, we present a new model $\mathcal{M}_h$ which modifies the previous one by introducing the holds[1] of the vessels into it. As holds are introduced in the model, $\mathcal{M}_h$ is now based on *Dynamic QCAP* model [52].

Dynamic QCAP assigns QCs to the holds of the vessels. Thus, once all the movements of one hold are done, the QC can move to another location (another hold in the same vessel or to other vessel). The notation, assumptions and constraints mentioned above remain valids, except for obtaining the handling time corresponding to each vessel. The handling time will be explained later. The optimization function is also the same as in Equation 3.4.

Following, we introduce some more notation:

- $h(V_i)$: Number of holds into V_i. All vessels have the same length hold.
- $c_j(V_i)$: Number of movements to load or unload containers from/into the hold j, $1 \leq j \leq h(V_i)$.
- $st_j(V_i)$: Starting time of working the QC j, $1 \leq j \leq q(V_i)$. Just one QC can be assigned to one hold.
- $ht_j(V_i)$: Handling time of the QC j, $1 \leq j \leq q(V_i)$.
- $h_j(V_i)$: Set of handling times of each hold assigned to the QC j, $1 \leq j \leq q(V_i)$.

Within the Container Terminals two other key factors are the ratio of berth usage (B_u) and the quay cranes throughput (T_{qc}). Berth usage is obtained by means of the Equation 3.9. It reflects the area held by vessels with respect to the maximum area. The maximum area depends on the length of the quay (L) as well as the fisrt mooring and the last departure of the incoming vessels.

$$firstArrival \leftarrow \min_i(m(V_i)) \tag{3.6}$$

$$lastDeparture \leftarrow \max_i(m(V_i)) \tag{3.7}$$

$$B_u \leftarrow \sum_i l(V_i) \times (d(V_i) - m(V_i)) \tag{3.8}$$

$$B_u \leftarrow \frac{B_u}{L \times (lastDeparture - firstArrival)} \tag{3.9}$$

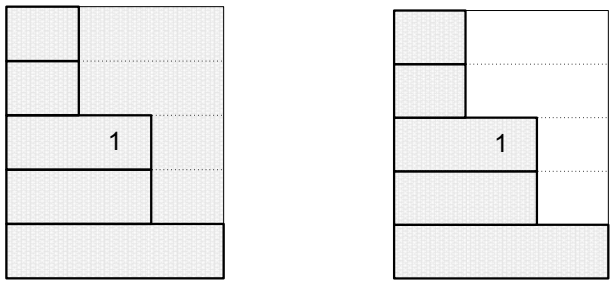
T_{qc} depends on the model for the QCAP we consider. Static QCAP model calculates T_{qc} by means of Equation 3.10 taking into consideration that one QC remains at the same vessel until it departs. Thereby, all the QCs are busy the same time although they are idle (shaded area in Figure 3.6(a)).

[1] The holds are the spaces of the container vessels where containers are stored.

$$T_{qc} \leftarrow \sum_{i} q(V_i) \times (d(V_i) - m(V_i)) \qquad \text{static} \qquad (3.10)$$

$$T_{qc} \leftarrow \sum_{i} \sum_{1 \leq j \leq q(V_i)} ht_j(V_i) \qquad \text{dynamic} \qquad (3.11)$$

However, Dynamic QCAP model, which uses Equation 3.11, considers that once one QC finishes its job at one hold, it can move to another location. Therefore, the time for each QC is only its service time (shaded area in Figure 3.6(b)).



(a) Static QCAP (b) Dynamic QCAP

Figure 3.6: Differences to calculate T_{qc}

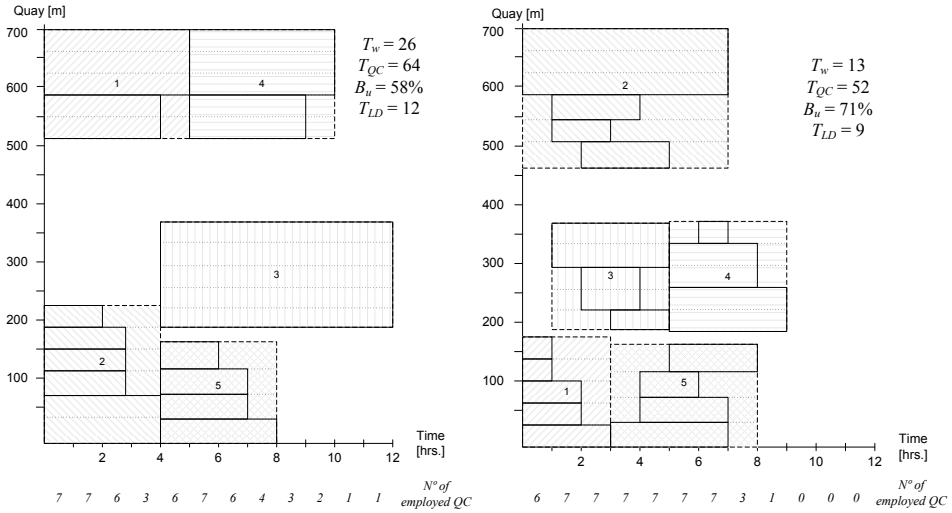
The fact of taking into consideration the holds ($h(V_i)$) of each vessel in our model allows better use of the resources (QCs and berth). Figure 3.7 shows both approximations. In this figure, each bold rectangle represents the time that one QC is working on a hold.

On the one hand, when one QC is assigned to one vessel i in Figure 3.7(a), this can not be moved to another vessel j until i leaves the Container Terminal. On the other hand, in Figure 3.7(b) the concept of holds is introduced. Once a QC finishes its job related to one hold in the vessel i, it could keep working on another hold of the same vessel or move to another vessel j. Thereby, the latter scheduling gets a departure time of the last vessel (T_{LD}) earlier than the former scheduling (from the unit time 12 to 9); the waiting time (T_w) is also reduced (from 26 to 13) and the berth usage rate (B_u) is increased as well (around 20%). Finally, Dynamic QCAP gets that more QCs are used per unit time than static QCAP, and the total time that the QCs are tied to one vessel (T_{qc}) is also reduced.

3.4 Mooring one vessel

Once all the parameters are defined, we present the function *moorVessel* (Algorithm 12) used to get moored one vessel V_i in a given time t (the required data are: v: Vessel for allocating; V_{in}: set of vessels already moored). These algorithms are written following the $\mathcal{M}_h$ model (Section 3.3.2).

In this mooring process, three steps are distinguished (Figure 3.8):



(a) QCs assigned to one vessel (b) QCs assigned to holds of one vessel

Figure 3.7: Approximation using or not the holds of the vessels

1. Check if there are available QCs during the handling time of V_i (Algorithm 13).

2. Make sure there is enough continuous length at the berth to moor V_i (Algorithm 15).

3. Assign more cranes when it is possible (Algorithm 16) (only for dynamic model, Section 3.3).

The main differences between the two models described in previous sections are:

- how the handling time is calculated. Algorithms for the $\mathcal{M}_v$ model are described in [53]; and

- $\mathcal{M}_v$ does not carry out the adding cranes phase.

Algorithm 13 presents how to obtain the number of QCs. As we have mentioned above, in this chapter we consider that each QC is assigned just to one hold. In order to do this, we distribute the holds of one vessel ($h(V_i)$) into the different cranes by the Algorithm 14.

After determining this first number of QCs, we must check if there is enough continuous length (Algorithm 15). At this moment, the distance security length between two contiguous vessels is taken into account (`secQC`). Furthermore, the given berthing position will be as close to the ends of the berth as possible.

Then, if the vessel V_i has available QCs and length to get moored at the time t, it is tried to assign more QCs. This process is carried out in Algorithm 16. This is based on obtaining the period of time in which there is at least one available QC

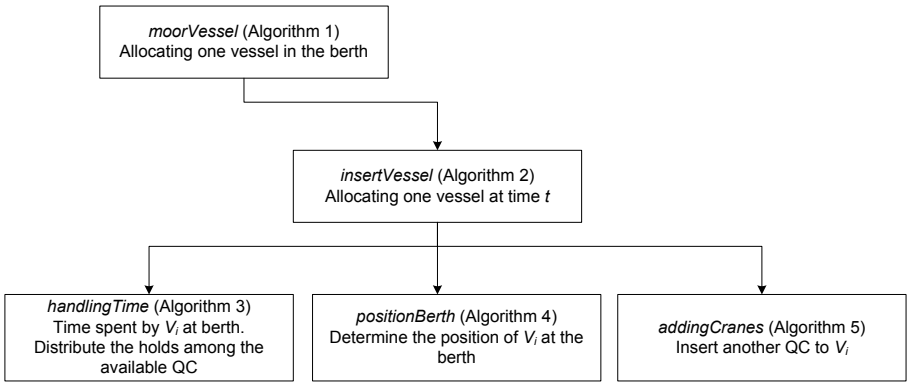


Figure 3.8: Application order of the algorithms presented

Algorithm 12: Function moorVessel. Allocating exactly one vessel in the berth.

Data: v vessel to moor; V_{in} elements.
if $|V_{in}| = 0$ **then**
  $cranes \leftarrow \max\left(1, \min\left(\texttt{maxQC}, \texttt{floor}\left(\frac{l(v)}{\texttt{secQC}}\right)\right)\right)$;
  $t_f \leftarrow t + \texttt{handlingTime}(v, nc)$;
  $m(v) \leftarrow t$;
  $d(v) \leftarrow t_f$;
  $q(v) \leftarrow cranes$;
  $pos(v) \leftarrow 0$;
  $V_{in} \leftarrow V_{in} \cup v$;
else
  $inst \leftarrow insertVessel(v, a(v), V_{in})$;
  if $!inst$ **then**
    $T \leftarrow st_k(v_j) + ht_k(v_j) \mid v_j \in V_{in}, 1 \leq k \leq q(v_j) \ \wedge \ st_k(v_j) + ht_k(v_j) > a(v)$;
    while $t_k \in T \wedge !inst$ **do**
      $inst \leftarrow insertVessel(v, t_k, V_{in})$;
    end
  end
end

from $m(V_i)$ until $d(V_i)$, and then setting a new QC to the ship if there is any hold in which there is no QC working yet. Thus, we can reduce the departure time of this vessel.

Finally, if the vessel V_i can not be moored at this time t, the whole process described above is repeated taking into consideration other time (t_k) to moor V_i (Algorithm 12). Each time t_k represents each time one crane has finished working on the hold of the others vessels.

3.5 A meta-heuristic method for BAP+QCAP

From the methods explained above, we have developed different methods for solving BAPs. Firstly, we applied the simplest solution for both $\mathcal{M}_v$ and $\mathcal{M}_h$ models,

Algorithm 13: Function insertVessel. Allocating one vessel in the berth at time t

Data: V_i: Vessel for allocating; t: actual time; V_{in}: moored already vessels;
Result: V_i could moor;

$L_{avail} \leftarrow L - \left(\sum_{v_j \in V_{in}} l(v_j) \mid m(v_j) \leq t \wedge d(v_j) > t\right)$;
if $L_{avail} \leq l(V_i)$ **then**
  return *false*;
end
$cranes \leftarrow -1$;
$cranes_m \leftarrow -1$;
repeat
  $nc \leftarrow \max(1, cranes)$;
  $t_f \leftarrow t + \texttt{handlingTime}(V_i, nc)$;
  $cranes_m \leftarrow nc$;
  $cranes \leftarrow \max\left(1, \min\left(\texttt{maxQC}, \texttt{floor}\left(\frac{l(V_i)}{\texttt{secQC}}\right)\right)\right)$;
  /* Vessels which coincide with V_i */
  $W \leftarrow v \in V_{in} |\ d(v) > t \wedge m(v) < t_f$;
  $QC_u \leftarrow \texttt{cranesWorking}(v, t), \forall v \in W$;
  $cranes \leftarrow \min(cranes, QC - QC_u)$;
  foreach $i \in W$ **do**
    if $m(i) \geq t$ **then**
      $QC_u \leftarrow \texttt{cranesWorking}(v, m(i)), \forall v \in W$;
      $cranes \leftarrow \min(cranes, QC - QC_u)$;
    end
    for $j \leftarrow 1$ **to** $q(i)$ **do**
      $QC_u \leftarrow \texttt{cranesWorking}(v, st_j(i) + ht_j(i)), \forall v \in W$;
      $cranes \leftarrow \min(cranes, QC - QC_u)$;
    end
  end
  if $cranes \leq 0$ **then**
    return *false*;
  end
until $cranes_m = cranes$;
$q(V_i) \leftarrow cranes_m$;
$m(V_i) \leftarrow t$;
$d(V_i) \leftarrow t_f$;
$insert \leftarrow \texttt{PositionBerth}(V_i, V_{in})$;
if $insert$ **then**
  $\texttt{addingCranes}(V_i, V_{in})$;
end
return $insert$;

Algorithm 14: Function handlingTime. Distribute the holds among the available QC

Data: V_i: Vessel to allocate; nQC: number of QC;

$T \leftarrow c_j(V_i)/$ `movsQC`, $1 \leq j \leq h(V_i)$;
`/* Sort the values of` T `from top to bottom */`
$q(V_i) \leftarrow nQC$;
for $j \leftarrow 1$ **to** nQC **do**
    $st_j(V_i) \leftarrow m(V_i)$;
    $ht_j(V_i) \leftarrow$ `ceil`(T_j);
    $h_j(V_i) \leftarrow \{$`ceil`$(T_j)\}$;
end
for $j \leftarrow nQC + 1$ **to** $h(V_i)$ **do**
    `/* Choose the QC which ends earlier */`
    $q_m \leftarrow$ `argmin`$(st_j(V_i) + ht_j(V_i),\ 1 \leq j \leq nQC)$;
    $h_{q_m}(V_i) \leftarrow h_{q_m}(V_i) \cup T_j$;
    $ht_{q_m}(V_i) \leftarrow ht_{q_m}(V_i) +$ `ceil`(T_j);
end
return $\max(st_j(V_i) + ht_j(V_i)),\ 1 \leq j \leq q(V_i)$;

following the FCFS criteria: $\forall i, m(V_i) \leq m(V_{i+1})$. A vessel can be allocated at time t when there is no vessel moored in the berth or there are available contiguous quay length and cranes at time t (Algorithm 17).

Just for the $\mathcal{M}_v$ model (Section 3.3), we also have implemented a complete search algorithm for obtain the best (optimal) mooring order of vessels: the lowest T_w (lower bound of the cost function). This algorithm uses the functions *moorVessel* (Algorithm 12) to allocate one vessel from its arrival time (Section 3.4). The lowest T_w is obtained by checking each possible order of the incoming vessels.

The next developed method is a meta-heuristic GRASP algorithm for Berth Allocation and Quay Crane Assignment Problem (Algorithm 18). This method has been developed for both $\mathcal{M}_v$ and $\mathcal{M}_h$ models. This is a randomly-biased multistart method to obtain optimized solutions of hard combinatorial problems in a efficient way. The parameter δ ($0 \leq \delta \leq 1$) allows tuning of search randomization.

This algorithm receives as parameters both the δ factor and the set of vessels V_{out} waiting for mooring at the berth. Firstly, all the waiting vessels V_{out} are considered as candidates C. Each of the candidate vessels is moored within the current state (being assigned the mooring and departure times ($m(V_i)$,$d(V_i)$), the number of QCs ($q(V_i)$) and the berthing position ($pos(V_i)$)) and they are valued according to the cost function f_c. This cost function is the sum of T_w that each vessel causes to the rest of unmoored vessels.

According to the cost function f_c, a restricted candidate list (RCL) is created. Then, one vessel v is chosen to be definitely moored by following the random degree indicated by δ factor. Once v is determined, this is added to the set of vessels V_{in} and eliminated from the candidate list C. This loop is repeated until C is empty, that is, all the vessels are moored.

As metaheuristic GRASP indicates, this search is repeated according to the number of iterations specified by the user. Thus, the best solution according to T_w

Algorithm 15: Function `PositionBerth`. Determining the position of the vessel in the berth

Data: V_i: Vessel to allocate; V_{in}: moored vessels.
Result: V_i could moor (by length).
$W' \leftarrow v \in V_{in} |\ d(v) \leq m(V_i) \wedge m(v) \geq d(V_i)$;
$W \leftarrow V_{in} - W'$;
`sortByPositionBerth` (W);
`/* Position occupied by the moored vessels */`
$busy = \{0\}$;
$lengths = \{0\}$;
foreach $v \in W$ **do**
    $busy = busy \cup \{pos(v)\}$;
    $lengths = lengths \cup \{l(v)\}$;
end
$busy = busy \cup \{L\}$;
$lengths = length \cup \{0\}$;
$minDistance \leftarrow +\infty$;
$posBerth \leftarrow 0$;
$assigned \leftarrow false$;
`/* Examining each gap between the vessels */`
for $i \leftarrow 1$ **to** $|busy| - 1$ **do**
    $dLeft \leftarrow lengths[i] \times$ `secLength`;
    $dRight \leftarrow lengths[i+1] \times$ `secLength`;
    $dVessel \leftarrow l(V_i) \times$ `secLength`;
    $dLeft \leftarrow \max(dLeft, dVessel)$;
    $dRight \leftarrow \max(dRight, dVessel)$;
    $free \leftarrow (busy[i+1] - dRight) - (dLeft + (busy[i] + lengths[i]))$;
    `/* Choose the berthing position closer to the ends of the berth */`
    if $free \geq l(V_i)$ **then**
        $assigned \leftarrow true$;
        if $minDistance > (busy[i] + lengths[i] + dLeft)$ **then**
            $minDistance \leftarrow busy[i] + lengths[i] + dLeft$;
            $posBerth \leftarrow minDistance$;
        end
        if $minDistance > (L - (busy[i+1] - dRight))$ **then**
            $minDistance \leftarrow L - (busy[i+1] - dRight)$;
            $posBerth \leftarrow L - (minDistance + l(V_i))$;
        end
    end
end
if $assigned$ **then**
    $pos(V_i) \leftarrow posBerth$;
    return *true*;
end
return *false*;

Algorithm 16: Function addingCranes. Insert another QC to V_i

Data: V_i: Vessel to allocate; V_{in}: moored vessels;
// Set of vessels which are moored at the same time than V_i
$W \leftarrow v \in V_{in} \mid d(v) > a(V_i) \ \wedge \ m(v) < d(V_i)$;
// T is the set of berthing and ending times of each QC from W
$T \leftarrow m(v) \mid v \in W \ \wedge \ m(v) \geq t$;
$T \leftarrow T \cup \{st_j(v) + ht_j(v) \mid v \in W, \ 1 \leq j \leq q(V_i) \ \wedge \ (st_j(v) + ht_j(v)) < m(V_i) \wedge$
$(st_j(v) + ht_j(v)) \geq d(V_i)\}$;
// The values of T must be sorted from bottom to top
// Obtain the time $[start, end]$ that at least there is 1 available QC
$continue \leftarrow false$ **repeat**
  $start \leftarrow -1$; $end \leftarrow -1$;
  foreach $t \in T$ **do**
    $Q_u \leftarrow$ cranesWorking$(v, t), \forall v \in W$;
    if $|Q_u| > 0$ **then**
      if $start = -1$ **then**
        $start \leftarrow t$; $end \leftarrow -1$;
      end
    else
      if $start \neq -1$ **then**
        $end \leftarrow t$;
      end
    end
    if $start \neq -1 \ \wedge \ end \neq -1$ **then**
      // Find a hold whose handling work is lower than $end - start$
      // $last(h_j(V_i))$ is the last hold assigned to this QC
      // Handling time of the hold
      $H \leftarrow \max(last(h_j(V_i))), 1 \leq j \leq q(V_i) \ \wedge \ last(h_j(V_i)) < (end - start)$;
      // QC which carries out this hold
      $k \leftarrow$ argmax$(last(h_j(V_i))), 1 \leq j \leq q(V_i) \ \wedge \ last(h_j(V_i)) < (end - start)$;
      // QC which finishes at $start$ time
      $m \leftarrow j \mid 1 \leq j \leq q(V_i) \ \wedge \ st_j(V_i) = start$;
      if $H \neq \varnothing$ **then**
        $continue \leftarrow true$;
        // Delete the hold from the first QC
        $h_k(V_i) \leftarrow h_k(V_i) - H$;
        $ht_k(V_i) \leftarrow ht_k(V_i) - H$;
        // Add this hold to the new QC
        if $m \neq \varnothing$ **then**
          $k \leftarrow m$; // Choosing a QC already assigned to V_i
          $ht_k(V_i) \leftarrow H$;
          $h_k(V_i) \leftarrow h_k(V_i) \cup \{H\}$;
        else
          if $|h_k(V_i)| > 0$ **then**
            $k \leftarrow q(V_i) + +$; // A new QC is moved to V_i
          end
          $st_k(V_i) \leftarrow start$;
          $ht_k(V_i) \leftarrow ht_k(V_i) + H$;
          $h_k(V_i) \leftarrow \{H\}$;
        end
      end
      $d(V_i) \leftarrow \max(st_j(V_i) + ht_j(V_i)), 1 \leq j \leq q(V_i)$;
      $start \leftarrow -1$; $end \leftarrow -1$;
    end
  end
until $!continue$;

Algorithm 17: Allocating vessels following FCFS policy

Data: V: set of ordered incoming vessels;
Result: Sequence for V;
$V_{last} \leftarrow \varnothing$;
$V_m \leftarrow \varnothing$;
foreach $V_i \in V$ **do**
  $t \leftarrow \max(m(V_{last}), a(V_i))$;
  $inst \leftarrow insertVessel(V_i, t)$;
  if $!inst$ **then**
    $T \leftarrow st_k(v_j) + ht_k(v_j) \mid v_j \in V_{in}, 1 \leq k \leq q(v_j) \ \wedge \ st_k(v_j) + ht_k(v_j) > t$;
    while $t_k \in T \wedge !inst$ **do**
      $inst \leftarrow insertVessel(V_i, t_k)$;
    end
  end
  $V_{last} \leftarrow V_i$;
  $V_m \leftarrow V_m \cup V_i$;
end

Algorithm 18: Grasp metaheuristic adapted to BAP

Data: δ factor; V_{out} elements; b: state of the berth
Result: Solution s
$V_{in} \leftarrow \{\}$;
$C \leftarrow V_{out}$;
while $C \neq \varnothing$ **do**
  foreach $v_e \in V_{out}$ **do**
    $f_c(v_e) \leftarrow 0$;
    $b' \leftarrow b$;
    $moorVessel(v_e, V_{in})$;
    $V'_{in} \leftarrow V_{in} \cup v_e$;
    foreach $v_o \in V_{out} \mid v_o \neq v_e$ **do**
      $moorVessel(v_o, V'_{in})$;
      $f_c(v_e) \leftarrow f_c(v_e) + (w(v_o) \times pr(v_o))$;
    end
  end
  $c_{inf} \leftarrow \min\{f_c(e) \mid e \in C\}$;
  $c_{sup} \leftarrow \max\{f_c(e) \mid e \in C\}$;
  $RCL \leftarrow \{e \in C \mid f_c(e) \leq c_{inf} + \delta(c_{sup} - c_{inf})\}$;
  $v \leftarrow random(RCL)$;
  $moorVessel(v, V_{in})$;
  $update(b)$; `/* state of the berth` b `with` v `moored */`
  $V_{in} \leftarrow V_{in} \cup v$;
  $C \leftarrow C - v$;
end

is returned as the solution for the BAP.

This metaheuristic process does not include a local search technique since it would involve testing the possible exchanges between the already ordered vessels, so that the computational cost would be increased considerably.

3.6 Evaluation

In this section, we evaluate the behavior of the three methods presented. The experiments were performed on instances given by the port operators. These instance are composed on 20 vessels with an exponential distribution of arrivals and all needed factors are fixed (length, draft and moves). Moreover, port operators gave us instances with two different inter-arrival distributions: *Spar* means that the arrival among vessels is sparsely distributed meanwhile *Dens* means that the arrival among vessels is densely distributed.

As we mentioned above, our goal is to minimize the total waiting time elapsed to serve the set of n vessels. All the experiments carried out were solved on a personal computer equipped with a Core 2 Quad Q9950 2.84Ghz with 3.25Gb RAM.

Considering the $\mathcal{M}_v$ model, (see table 3.1) the times of employing the complete search against the GRASP method with 1000 iterations is presented. As it can be observed, a complete search is impracticable for 13 vessels (approximately 7 hours). However, our GRASP method takes around 15 seconds to solve a schedule of 20 vessels.

Table 3.1: Computing time elapsed in milliseconds (ms.)

Number	**Complete**	**GRASP**
5	<1	203
10	38110	1486
11	334441	1734
12	2689831	2234
13	26063570	3219
14	$\cdots$	3907
15	$\cdots$	5203
20	$\cdots$	14954

Table 3.2 shows the average T_w (with $\gamma = 1$) of 100 instances using FCFS and complete search methods described above with two different inter-arrival distributions. Through these data, it can be observed that FCFS method results a schedule which is far away from the best one.

Table 3.2: T_w for sparsely/densenly distributed Vessels

Vessels	**FCFS**	**Complete**
5 - Spar	27.67	13.93
10 - Spar	91.06	43.07
5 - Dens	52.79	32.83
10 - Dens	274.00	156.61

The rest of the experiments will be based on $\mathcal{M}_h$ model. The next experiments compare our GRASP method against the FCFS criteria, using 100 instances with the former inter-arrival distributions. Figure 3.9 and Figure 3.10 show the average values for the metric function for the two methods to allocate 10 vessels. It can be observed that the solutions given by our GRASP method always outperformed the FCFS solution, mainly for $\delta \in [0.3 - 0.5]$ in densely distributed vessels and $\delta \in [0.3 - 0.4]$ for sparsely distributed ones.

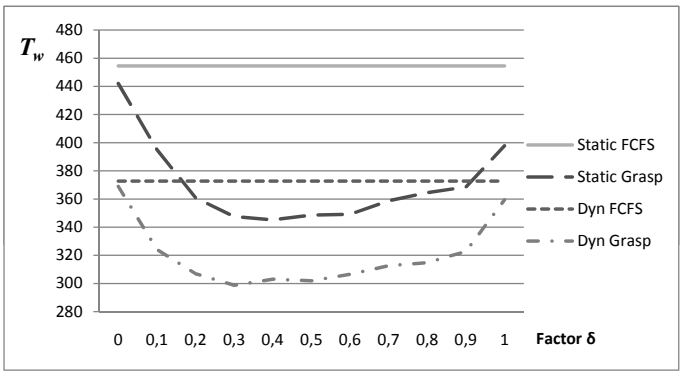


Figure 3.9: T_w for 10 vessels with densely distributed inter-arrival times.

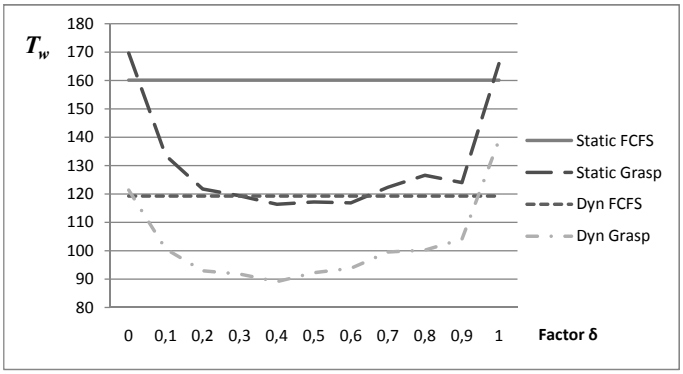


Figure 3.10: T_w for 10 vessels with sparsely inter-arrival times.

Furthermore, Figure 3.11 shows the T_w values for 10 incoming vessels. It can be observed as the number of iterations increased the quality of our GRASP method also increased. For instance, for $\delta = 0.4$, $T_w = 303.03$ with 100 iterations meanwhile T_w is decreased to 280.8 with 400 iterations.

In Figure 3.12 and Figure 3.13, the same evaluation is carried out for 20 vessels. It can be observed the same tendency than in Figure 3.9 and Figure 3.10, where $\delta \in [0.1 - 0.2]$ got the lowest values for both inter-arrival time distributions. Moreover, GRASP reduces around $20 - 30\%$ the averages values of T_w with respect to the obtained by the FCFS criteria.

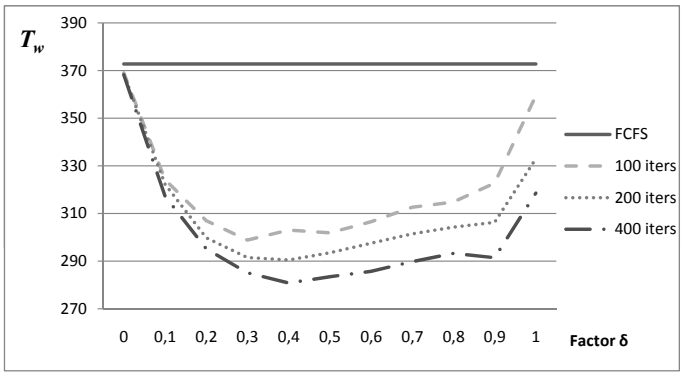


Figure 3.11: T_w depending on the number of iterations in GRASP

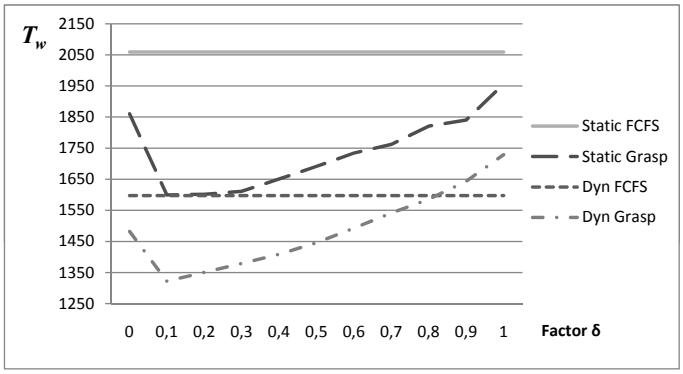


Figure 3.12: T_w for 20 vessels with densely inter-arrival times.

The last experiments also show how the Dynamic QCAP model always outperforms the static one. For instance, taking into account 20 incoming vessels and $\delta = 0.2$ in Figure 3.13, the value of T_w is 255.43 and 374.87 for dynamic and staic QCAP model respectively.

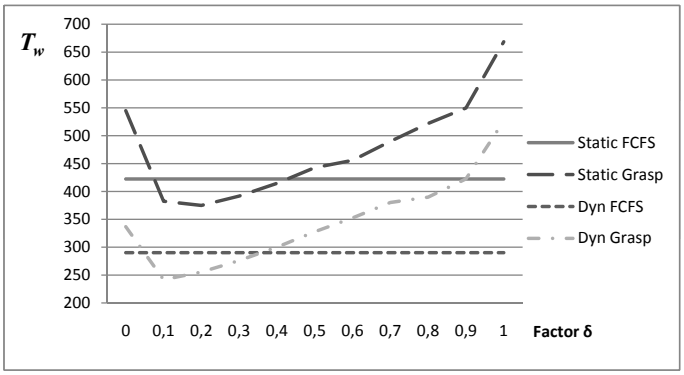


Figure 3.13: T_w for 20 vessels with sparsely inter-arrival times.

Figure 3.14 shows the evolution of FCFS when the number of incoming vessels with the static QCAP (solutions that would be provided by terminal operators) against GRASP considering Dynamic QCAP solutions. It is remarkable, the greater the number of incoming vessels is, the better the GRASP solutions.

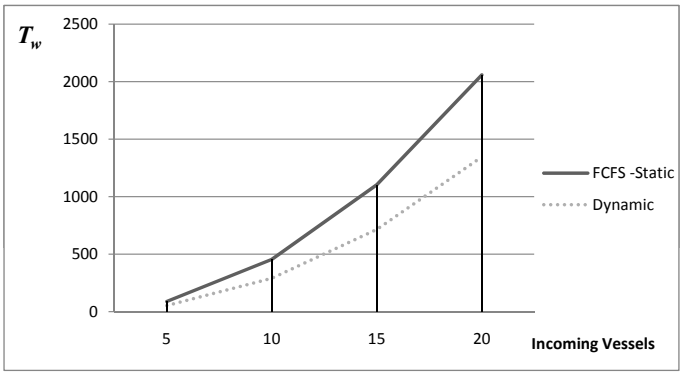
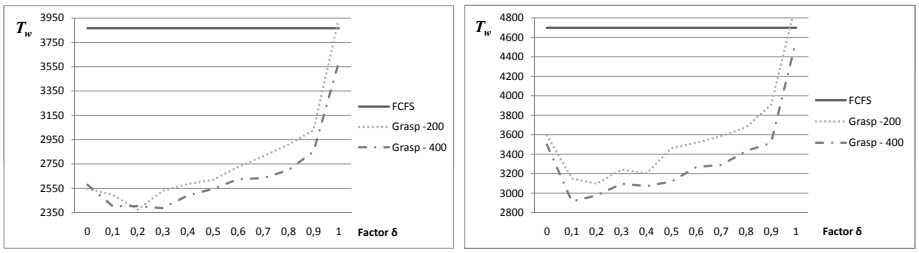


Figure 3.14: T_w varying the number of incoming vessels.

This metaheuristic search has been also applied to real-data given by port operators from MSC. Figure 3.15 shows that our grasp metaheuristic achieves better results than FCFS method as the examples studied above. Factor $\delta \in [0.1 - 0.2]$ gets the best T_w for 15 incoming vessels, and in the case of 10 vessels the best factor is $\delta \in [0.3 - 0.4]$.

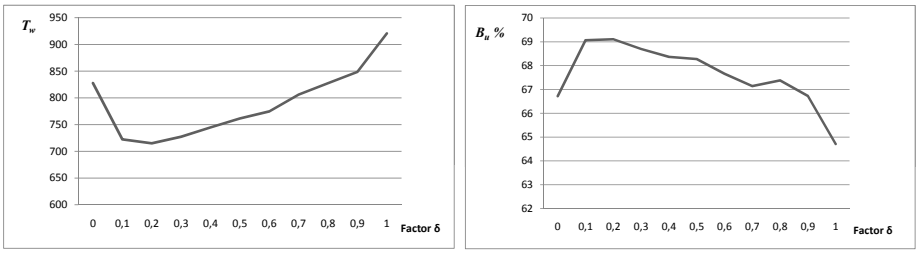


(a) Dynamic QCAP model (b) Static QCAP model

Figure 3.15: T_w for real-data obtained given by port operators.

Following the same experiments as mentioned before, Figure 3.16 shows the relationship between berth usage and the weighted waiting time T_w taking into account 15 incoming vessels. We can directly confirm that the lower T_w is, the greater berth usage.

Finally, Table 3.3 shows that considering holds in the model (Dynamic QCAP), the throughput of the QCs is improved considerably. In other words, QCs spend less time in order to do the same amount of movements. Therefore, Dynamic QCAP model allows better use of the QC, since they can be used in other vessels immediately.



(a) Weighted waiting time T_w (b) Berth usage ratio B_u

Figure 3.16: Relationship between ratio of berth usage and T_w (15 incoming vessels)

Table 3.3: Average time of the QC that they are busy (15 vessels with densely inter-arrival times)

Number of Vessels	**Static QCAP**	**Dynamic QCAP**
10	233.85	202.02
15	351.15	303.08
20	453.44	406.68

3.7 Conclusions

In this chapter, we present a new process to allocate berth space for a number of ships using a GRASP metaheuristic. This process also adds the Quay Crane Assignment Problem into a model that takes into account the holds of each incoming vessel. This method has been compared to the actual scheduling method employed in Container Terminals. It is observed how it can reduce the waiting time, the berth utilization and throughput of QCs. Thereby, we state that the method used in Container Terminals can be improved by using our different metaheuristics, for instance GRASP.

The above experiments show that:

- the greater the number of vessels to schedule is, the more significant the difference between GRASP and FCFS criteria, and
- the berth usage is directly proportional to total weighted waiting time (T_w). It means, the lower the T_w is, the lower the berth usage.

In this chapter, we also present a model where the basic unit to assign QCs is each one of the holds of the incoming vessels. This model gets better results than the former model, whose basic unit is the own vessel. The reason is that QCs can be employed in an efficient manner, moving each time they finish their job at one hold.

Chapter 4

Integration BAP, QCAP and CStackP

4.1 Introduction

The efficient management of containers in port requires more analysis and development to ensure reliability, delivery dates or handling times in order to improve productivity and container throughput from quay to landside and vice versa. Extensive surveys are provided about operations at seaport container terminals and methods for their optimization [69, 66]. Moreover, other problems are faced on planning the routes for liner shipping services to obtain the maximal profit [9]. Another important issue for the success at any container terminal is to forecast container throughput accurately [6]. Thus, they could develop better operational strategies and investment plans.

The main research on optimization methods in container terminals are related to reduce the berthing time of vessels. This objective generates a set of interrelated problems such as berth allocation, yard-side operation, storage operation and gate-house operation. Usually, each one of these problems is managed independently of others due to their exponential complexity. However, these problems are clearly interrelated so that an optimized solution of one of them restrings the possibility of obtaining a good solution in another.

In this chapter, we focus our attention on three important and interrelated problems: the Berth Allocation Problem (BAP), the Quay Crane Assignment Problem (QCAP) and the Container Stacking Problem (CStackP) (see Figure 4.1). Briefly, the BAP and QCAP consist on the allocation of docks and quay cranes to incoming vessels under several constraints and priorities (length and depth of vessels, number of containers, and so on). On the other hand, when a vessel berths, export containers stacked to be loaded in the vessel should be on top of the stacks of the container yard. Therefore, the CStackP consists on relocating the containers so that the yard crane does not need to do re-handling work at the time of loading. These two problems are clearly related: an optimal berth allocation plan may generate a

large amount of relocations for export containers; meanwhile a suboptimal berth allocation plan could require fewer rearrangements.

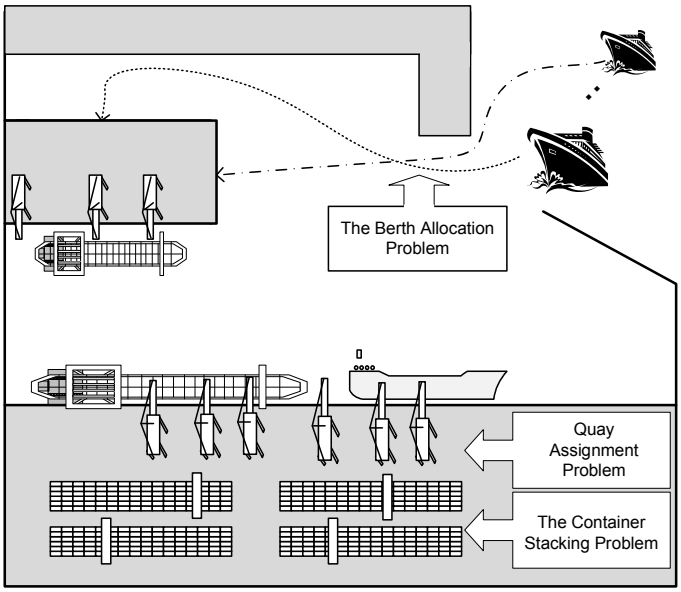


Figure 4.1: Integrated Remarshalling, Berthing and Quay Crane allocation problems in Maritime Terminals.

In order to provide a computer-based decision support system, we integrate a set of intelligent techniques for solving these problems concurrently in order to achieve a mixed-solution that combines optimization of BAP, QCAP and CStackP [57, 56, 61, 62]. To this end, we integrate the solutions obtained by the developed methods in previous chapters:

- the heuristically-guided planner for generating a rehandling-free intra-block remarshalling plan for container yards (CStackP problem) presented in Chapter 2; and

- the GRASP metaheuristic approach for solving the BAP and QCAP as an independent problem (Chapter 3).

With this data, terminal operators should ultimately decide in each scenario which solution is the most appropriate in relation to a multi-objective function: to minimize the waiting times of vessels and to minimize the amount of relocations of containers.

These techniques will be very useful for terminal operators due to berth allocation is especially important in case of ship delays. A new berthing place has to be allocated to the ship whereas containers are already stacked in the yard [66] and a remarshalling plan remains necessary to minimize the berthing time.

4.2 Integrating BAP, QCAP and CStackP

As we have pointed out, both the CStackP and the BAP+QCAP are well-known problems and several techniques have been developed to solve them in an independent way. However, few systems have been developed to relate and optimize both problems in an integrated way. Some works consider berth and yard planning in a common optimization model [1][5][14], but they are mainly focused on storage strategies. Moreover, only some works integrate the BAP with the QCAP. Giallombardo et al. [17] try to minimize the yard-related house-keeping costs generated by the flows of containers exchanged between vessels. However, there also exists a relationship between the optimization of maritime and terminal-sides operations (BAP, QCAP, CStackP, etc.).

Figure 4.2 shows an example of three berth allocation plans with the corresponding quay crane allocations and a block of containers to be loaded in the vessels. Containers of type A, B and C must be loaded in vessels A, B and C, respectively. In the first berth allocation plan, the order of vessel is A-B-C and the quay crane allocation is two cranes, three cranes and one crane, respectively. The second berth allocation plan is C-B-A. In this case the quay crane allocation is three, two and one, respectively. Finally, the third berth allocation plan is B-C-A and two quay cranes are allocated to all vessels. Each configuration generates a different waiting time for berthing and different handling times, and the port operator probably selects the best solution to optimize these (BAP and QCAP) problems. However the best solution of these two problems could generate a large number of reshuffles in the yard so the question is straightforward: what is a better solution? Perhaps a solution that optimizes the BAP+QCAP could not be the more appropriate for the CStackP (and vice versa).

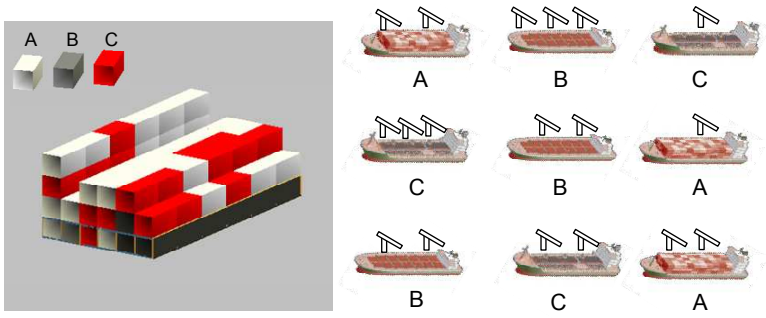


Figure 4.2: Different alternatives of BAP and QCAP.

Given a waiting queue of vessels to be allocated and a given state of the containers in the container yard, each solution for the BAP+QCAP ($SBAP_i$: a feasible sequence of mooring and $SQCAP_i$: a feasible quay crane allocation), requires a different number of container's relocations in the associated CStackP solution ($SCStackP_i$) in order to put on top the containers to be loaded according to the order of berthing. We can associate a cost to each $SBAP_i + QCAP_i$ related to

the total weighted waiting time and handling time of vessels of this berthing order (T_w). Likewise, we can associate a cost to each $SCStackP_i$ as the number of required container relocations. Therefore, we can qualify the optimality of each global solution (Sol_i) of BAP+QCAP and CStackP as a lineal combination of the quality of each partial solution:

$$Cost(Sol_i) = \alpha * Cost(SBAP_i + SQCAP_i) + \beta * (SCStackP_i) \qquad (4.1)$$

The best decision will depend on the policy of each maritime terminal (α and β parameters). The data flow diagram of the Integrated System Functioning can be seeing in Figure 4.3. Firstly, the BAP, QCAP and the CStackP data are loaded in the integrated system. Next, the BAP+QCAP is solved to achieve a solution ($SBAP_i + QCAP_i$) based on their constraints and optimization criteria. Then, the CStackP is estimated by taken into account the berthing order of vessels obtained in $SBAP_i + QCAP_i$. This estimator returns the number of reshuffles needed to achieve a solution. After this step, the cost of the global solution (Sol_i) can be calculated by using the previous expression 4.1. By iterating this integrated process, the operators can obtain a qualification cost of each feasible Sol_i, as well as the best global solution, according the given α and β parameters. A branch and bound method has been also applied in the integrated search for the best global solution (Sol_i), so that the search can be pruned each time the current solution does not improve the best solution found so far. Finally, once the best solution is obtained, the CStackP planner is carried out to obtain the specific movements for all remarshalling tasks. This plan is sequentially obtained for each vessel according to the solution obtained in $SBAP_i + QCAP_i$ and the current state of the container yard. Thus, the optimized remarshalling plan for the berthing order of vessels of $SBAP_i$ is obtained.

After defining the Container Stacking Problem in Chapter 2 and Berth Allocation Problem together with Quay Crane Assingment Problem in Chapter 3 we can achieve a global solution Sol_i as it appears in Figure 4.3.

4.3 Evaluation

In this section, we evaluate the behavior of the integrated system in random instances. We randomly generated scenarios consisting of a set of 10 vessels for berthing. Each vessel must load C containers randomly from the container yard. The yard was composed of 170 containers so that it remained empty once the vessels were loaded. For each problem, we generated 10 random instances with a different configuration of containers in the yard.

For this first experiment, we applied the Algorithm 18 to obtain the best 10 schedules for the incoming vessels. To solve the Container Stacking Problem, the domain-dependent planner with the heuristic (Algorithm 1) was employed.

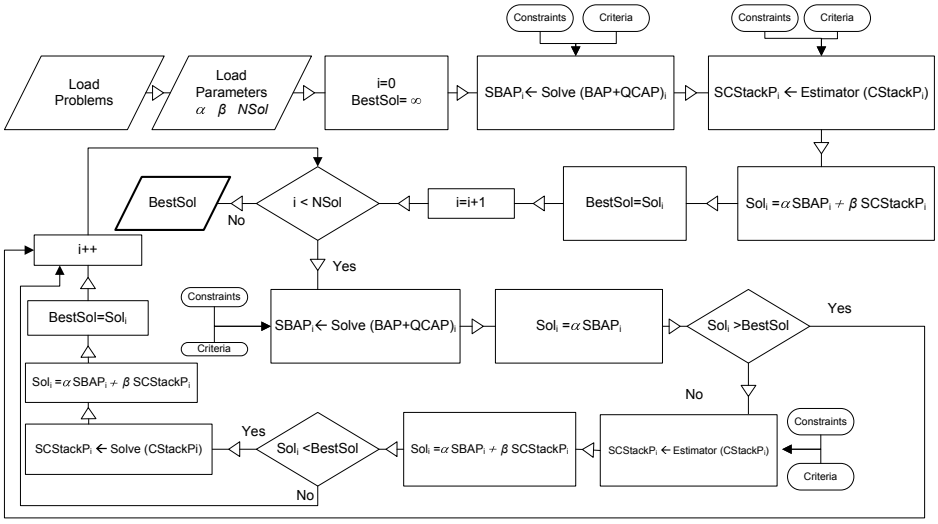


Figure 4.3: Data flow diagram of the Integrated System Functioning.

Figure 4.4 shows the combined function cost Cost(Sol_i), introduced in Equation 4.1 which relates for ten different scenarios:

- The normalized total weighted waiting time of vessels, Cost($SBAP_i$), and
- The number of its required container relocations, Cost($SCStackP_i$).

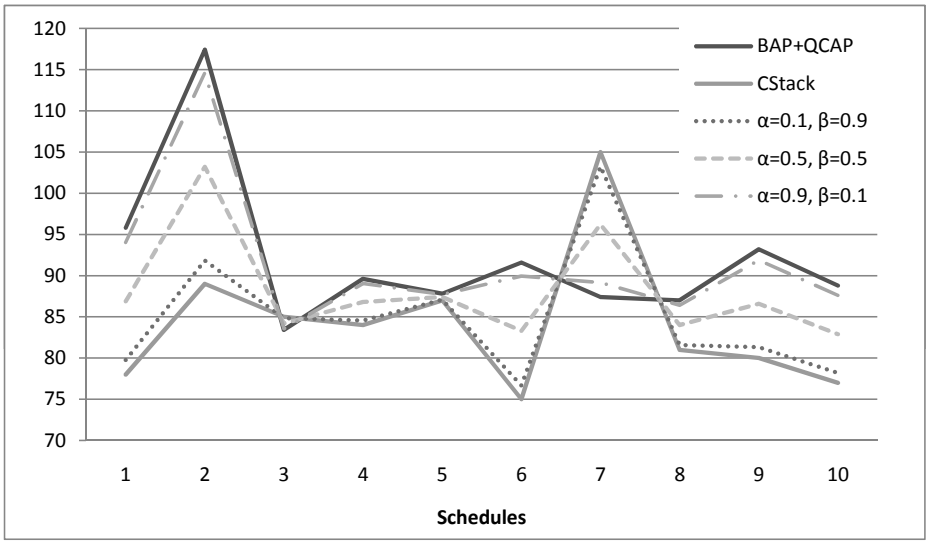


Figure 4.4: Relating the costs of BAP+QCAP and CStackP.

In each one of this ten cases, the arrival times and data of vessels, as well as the initial state of the container yard, have been randomly generated. Figure 4.4

represents the combined function cost, Cost(Sol_i) with three different weights of the parameters α and β. We can see that better (or worst) berthing orders can require larger (or smaller) number of container relocations. For instance, with $\alpha = 0.5, \beta = 0.5$ the best choice is the sixth schedule. It does not get the best solution for BAP+QCAP, however it corresponds to the schedule with the smallest number of container relocations (CStackP).

4.4 Conclusions

The Container Stacking Problem and the Berth Allocation Problem with the Quay Crane Assignment Problem are three important and related problems in maritime container terminals. In this chapter, we have presented an integrated system to manage these problems in a coordinated way. To this end, the developed solutions in the last chapters for each one of the problems are combined in our integrated system. Thus, terminal operators can be assisted to decide the most appropriated solution in each particular case. Furthermore, the system presented in this chapter could assist container terminal's operators to simulate, evaluate and compare different feasible alternatives to the same state of the yard and arrival of the ships.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The increase in the traffic of containers in the world makes necessary the construction of larger vessels by liner shippings. Therefore, Container Terminals must offer competitive services to ensure the lowest berthing time for these vessels. Because it is considered that the time a vessel spends moored at the terminal is nonproductive.

Thus, in this Master Thesis several artificial intelligence techniques have been applied to achieve optimal solutions to different combinatorial problems of Container Terminals (Figure 1.7). We have presented a domain-dependent heuristic planner for the Container Stacking Problem, specifically for the remarshalling problem. Next, we have developed a new metaheuristic solution for the Berthing Allocation Problem and Quay Crane Allocation Problem. And finally, we presented how to interrelate these two approximations in order to achieve an optimized solution for both problems.

Firstly, our domain-dependent planner is able to reduce the number of movements needed to put all the outgoing containers at top of the stacks or under another outgoing containers. It is able to handle real-world requirements, for instance balanced stacks or the existence of dangerous containers. And, it can get solutions for a whole block of containers. Furthermore, since this process needs a lot of computation time, an analytical formula was derived from the data in order to estimate the number of reshuffles for a given state of the yard.

The developed model in order to schedule incoming vessels gets plans which are near optimal solutions. Our model is based on the metaheuristic Grasp which introduces a random factor to get optimized solutions. For this problem, there are two remarkable points:

- The greater the number of incoming vessels, the greater the benefits to this metaheuristic model.
- How the berth usage is directly proportional to the total weighted waiting time.

And finally, we pointed out how these three issues (BAP, QCAP and Container Stacking Problem) are interrelated and we proposed an architecture to solve them in an efficient way. The best solution for berthing the incoming vessels might not offer an ordering which minimizes the number of reshuffles of the container yards. Therefore, an optimized global solution is a trade-off between these different problems.

With these provided solutions could assist container terminal's operators to simulate, evaluate and compare different feasible alternatives to the same initial states. For instance, evaluating the fact of using a different number of Quay Cranes at the berth, varying the maximum number dedicated by vessel or assessing several stacks configurations as allowing stack more containers (4, 5 or more tiers).

Summarizing, the next contributions have been presented in this work:

- A domain-dependent planner guided by a heuristic for the Container Stacking Problem.

  In addition, we have added real-world requirements to this planner:

  - Leave the outgoing containres near truck position.
  - Allow different heights.
  - Dangerous containers.
  - Balance one yard-bay and the whole container blocks.

- A domain-dependent planner to solve container blocks efficiently according to their number of outgoing containers.

- An analytic formula to estimate the number of reshuffles of one yard-bay.

- Berth Allocation and Quay Crane Allocation Problems have been solved by a model based on GRASP metaheuristic.

  Two approaches are studied depending on how Quay Cranes are allocated:

  - To vessels, called *static QCAP*.
  - To holds of vessels, *dynamic QCAP*.

- An efficient system to integrate the problems related to the storage yard (remarshalling) and the quay side (loading/unloading containers).

5.2 Future work

Although there have been studied these problems in the literature, there are still some open problems related to the Container Terminals.

In this work, we addressed the BAP and QCAP problems, but another important issue where quay cranes are involved is the Quay Crane Scheduling Problem

(QCSP). QCSP consists on determining the sequence of unloading/loading tasks assigned to each Quay Crane so that the completion time of a vessel tasks is minimized.

Related to the Container Stacking Problem, remarshalling problem is an option when there is enough time in order to prepare the container yard for the next vessel. However, when it is not possible, this problem and also our planner, should be adapted to achieve that all the outgoing containers are loaded onto the vessel minimizing the number of reshuffles during the pickup operation.

Other problems that have not been studied in this work and could be faced are for instance planning the routes for liner shipping services to obtain the maximal profit [9], or routing all the manned and automated vehicles of a Container Terminal in a coordinated way, e.g. AGV, Quay Cranes and RMGs in order to avoid deadlocks.

This last issue, about the automated vehicles in Container Terminals, could also be studied as a part of a system. A system which involves the problems studied in this work (BAP, QCAP and CStackP) as well as the routing of these vehicles.

5.3 Related publications

In this section, it is shown a list of published publications to conferences and journals. For each one of them, JCR ranking (in case of journals) or CSC/CORE ranking (in case of conferences) is showed.

Journals

- M. A. Salido, M. Rodriguez-Molins and F. Barber.
  Integrated intelligent techniques for remarshalling and berthing in maritime terminals.
  Advanced Engineering Informatics, 2010 (JCR: 1.73).
  DOI: 10.1016/j.aei.2010.10.001

- M. A. Salido, M. Rodriguez-Molins and F. Barber.
  A Decision Support System for Managing Combinatorial Problems in Container Terminals.
  Journal Knowledge Based Systems, 2010 (JCR: 1.308).

- M. Rodriguez-Molins, M. A. Salido and F. Barber.
  Intelligent Planning for Allocating Containers in Maritime Terminals.
  Experts Systems with Application, (Submitted) 2010 (JCR: 2.908).

- M. Rodriguez-Molins, M.A. Salido, and F. Barber.
  A Metaheuristic Technique for Solving the Berth Allocation Problem.
  Engineering Applications of Artificial Intelligence, (Submitted) 2010 (JCR: 1.444).

- M. Rodriguez-Molins, M. A. Salido and F. Barber.
  A Metaheuristic Technique for Solving Seaside Port Problems.
  NETWORKS, (Submitted) 2010 (JCR: 1.213)

International Conferences

- M. A. Salido, O. Sapena, M. Rodriguez and F. Barber.
  A planning-based approach for allocating containers in maritime terminals.
  In 21st International Conference on Tools with Artificial Intelligence, ICTAI. Pages 567-571. IEEE, 2009. (CORE B; CSC Ranking: 0.74)

- M. Rodriguez-Molins, M. Salido and F. Barber.
  Domain-dependent planning heuristics for locating containers in maritime terminals.
  In The Twenty Third International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, IEA-AIE 2010. (CORE B, CSC: top 5 in Industrial Engineering Conference Ranking)

- M. A. Salido, M. Rodriguez-Molins and F. Barber.
  Towards the integration of berth allocation and container stacking problems in martime container terminals.
  In The International Conference on Harbor, Maritime & Multimodal Logistics Modelling and Simulation, HMS 2010.

- M. Salido, M. Rodriguez-Molins, and F. Barber.
  Artificial intelligence techniques for the integration of berth allocation and container stacking problems in container terminals.
  In AI-2010 Thirtieth SGAI International Conference on Artificial Intelligence, 2010 (CORE C).

National Conferences

- M. A. Salido, O. Sapena, M. Rodriguez, and F. Barber.
  A planning-based approach for allocating containers in maritime terminals.
  In CAEPIA'09: Thirteenth Spanish Conference for Artificial Intelligence, 2009.
  (CSC Ranking: 0.55)

- M. Salido, O. Sapena, M. Rodriguez, and F. Barber.
  Heuristicas dependientes del dominio en terminales de contenedores.
  In PSCS'09: Workshop for Planning, Scheduling and Constraint Satisfaction, 2009.

- M. Rodriguez-Molins, M. Salido, and F. Barber.
  A metaheuristic technique for solving the berth allocation problem.
  In VIII Jornadas de Aplicaciones y Transferencia Tecnológica de la Inteligencia Artificial, TTIA 2010 (AEPIA).

Bibliography

[1] M. aleb Ibrahimi, B. de Castilho, and C.F. Daganzo. Storage space vs handlingwork in container terminals. *Transportation Research-B*, 27B:13–32, 1993.

[2] K.R. Baker. *Elements of sequencing and scheduling*. Kenneth R. Baker, 1995.

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

[4] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[5] A. Bruzzone and R. Signorile. Simulation and genetic algorithms for ship planning and shipyard layout. *Simulation*, 71:74–83, 1998.

[6] S.H. Chen and J.N. Chen. Forecasting container throughputs at ports using genetic programming. *Expert Systems with Applications*, 37(3):2054 – 2058, 2010.

[7] C.Y. Cheong, K.C. Tan, and D.K. Liu. Solving the berth allocation problem with service priority via multi-objective optimization. In *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, pages 95 –102, 2 2009-march 30 2009.

[8] R. Choe, T. Park, M.S. Oh, J. Kang, and K.R. Ryu. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. *Journal of Intelligent Manufacturing*, pages 1–17, 2009. 10.1007/s10845-009-0273-y.

[9] T.N. Chuang, C.T. Lin, J.Y. Kung, and M.D. Lin. Planning the route of container ships: A fuzzy genetic approach. *Expert Systems with Applications*, 37(4):2948 – 2956, 2010.

[10] D.S. Consultants. Global container terminal operators annual review and forecast. *Annual Report*, 2010.

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

[12] R. Dekker, P. Voogd, and E. Asperen. Advanced methods for container stacking. *Container Terminals and Cargo Systems*, pages 131–154, 2007.

[13] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[14] L.M. Gambardella, A.E. Rizzoli, and M. Zaffalon. Simulation and planning of an intermodal container terminal. *Simulation*, 71:107–116, 1998.

[15] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20(1):239–290, 2003.

[16] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

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

[18] F. Glover and G.A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

[19] Carla P. Gomes. Artificial intelligence and operations research: challenges and opportunities in planning and scheduling. *Knowl. Eng. Rev.*, 15:1–10, March 2000.

[20] Y. Guan and R.K. Cheung. The berth allocation problem: models and solution methods. *OR Spectrum*, 26(1):75–92, 2004.

[21] M. Han, P. Li, and J. Sun. The algorithm for berth scheduling problem by the hybrid optimization strategy GASA. *Proceedings of the Ninth International Conference on Control, Automation, Robotics and Vision (ICARCV 2006)*, pages 1–4, 2006.

[22] P. Hansen, C. Ceyda Oguz, and N. Mladenovic. Variable neighborhood search for minimum cost berth allocation. *European Journal of Operational Research*, 191:636–649, 2008.

[23] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas. A visual programming system for automated problem solving. *Expert Systems with Applications*, 37(6):4611 – 4625, 2010.

[24] L. Henesey. Overview of Transshipment Operations and Simulation. In *Med-Trade conference, Malta, April*, pages 6–7, 2006.

[25] L.E. Henesey. *Multi-agent systems for container terminal management*. Citeseer, 2006.

[26] Y. Hirashima. A Q-learning system for group-based container marshalling with a-priori knowledge for ship loading. In *ICCAS-SICE, 2009*, pages 1728–1733. IEEE, 2009.

[27] Y. Hirashima, N. Ishikawa, and K. Takeda. A new reinforcement learning for group-based marshaling plan considering desired layout of containers in port terminals. In *Networking, Sensing and Control, 2006. ICNSC'06. Proceedings of the 2006 IEEE International Conference on*, pages 670–675. IEEE, 2006.

[28] J. Hoffman and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[29] Jörg Hoffmann. The metric-ff planning system: translating "ignoring delete lists" to numeric state variables. *J. Artif. Int. Res.*, 20(1):291–341, 2003.

[30] A. Imai, H.C. Chen, E. Nishimura, and S. Papadimitriou. The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(5):900–920, 2008.

[31] A. Imai, K.I. Nagaiwa, and TAT Chan Weng. Efficient planning of berth allocation for container terminals in Asia. *Journal of advanced transportation*, 31(1):75–94, 1997.

[32] Kalmar Industries. Kalmar container handling systems: complete range of products and knowhow., 2007.

[33] J. Kang, M.S. Oh, E. Ahn, K. Ryu, and K. Kim. Planning for intra-block remarshalling in a container terminal. In Moonis Ali and Richard Dapoigny, editors, *Advances in Applied Artificial Intelligence*, volume 4031 of *Lecture Notes in Computer Science*, pages 1211–1220. Springer Berlin / Heidelberg, 2006. 10.1007/11779568_128.

[34] Kap Hwan Kim. Evaluation of the number of rehandles in container yards. *Computers & Industrial Engineering*, 32(4):701 – 711, 1997. New Advances in Analysis of Manufacturing Systems.

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

[36] K.H. Kim and G.P. Hong. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33(4):940–954, 2006.

[37] K.H. Kim, Y.M. Park, and M.J. Jin. An optimal layout of container yards. *OR Spectrum*, 30(4):675–695, 2008.

[38] K.W. Kim, Y.M. Park, and K.R. Ryu. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124:89–101, 2000.

[39] S. Kirkpatrick, CD Gelatt, and MP Vecchi. Optimization by Simulated Annealing, Science, Vol. 220. *Number*, 4598:671–680, 1983.

[40] KK Lai and K. Shih. A study of container berth allocation. *Journal of Advanced Transportation*, 26(1):45–60, 1992.

[41] L.H. Lee, E.P. Chew, K.C. Tan, and Y. Han. An optimization model for storage yard management in transshipment hubs. *OR Spectrum*, 28(4):539–561, 2006.

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

[43] Y. Lee and N.Y. Hsu. An optimization model for the container premarshalling problem. *Computers & Operations Research*, 34(11):3295 – 3313, 2007.

[44] G. Mauri, A. Oliveira, and L. Lorena. A hybrid column generation approach for the berth allocation problem. *Eighth European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2008)*, 4972:110–122, 2008.

[45] P.J.M. Meersmans and R. Dekker. *Operations research supports container handling*. Econometric Institute, Erasmuns University Rotterdam, 2001.

[46] E. Nishimura, A. Imai, and S. Papadimitriou. Berth allocation planning in the public berth system by genetic algorithms. *European Journal of Operational Research*, 131:282–292, 2001.

[47] K. Park, T. Park, and K.R. Ryu. Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1098–1105. ACM, 2009.

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

[49] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Verlag, 2008.

[50] M. Rodriguez-Molins, M. Salido, and F. Barber. Domain-dependent planning heuristics for locating containers in maritime terminals. In *The Twenty Third International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems IEA-AIE 2010*, 2010.

[51] M. Rodriguez-Molins, M. Salido, and F. Barber. A metaheuristic technique for solving the berth allocation problem. In *VIII Jornadas de Aplicaciones y Transferencia Tecnológica de la Inteligencia Artificial, TTIA2010 (AEPIA)*, 2010.

[52] M. Rodriguez-Molins, M.A. Salido, and F. Barber. A Metaheuristic Technique for Solving Seaside Port Problems (Submitted). *NETWORKS (JCR: 1.213)*, 2010.

[53] M. Rodriguez-Molins, M.A. Salido, and F. Barber. A Metaheuristic Technique for Solving the Berth Allocation Problem (Submitted). *Engineering Applications of Artificial Intelligence (JCR: 1.444)*, 2010.

[54] M. Rodriguez-Molins, M.A. Salido, and F. Barber. Intelligent Planning for Allocating Containers in Maritime Terminals (Submitted). *Experts Systems with Application (JCR: 2.908)*, 2010.

[55] A. Rushton, P. Croucher, and P. Baker. *The handbook of logistics and distribution management*. Kogan Page Ltd, 2006.

[56] M. Salido, M. Rodriguez-Molins, and F. Barber. Artificial intelligence techniques for the integration of berth allocation and container stacking problems in container terminals. In *AI-2010 Thirtieth SGAI International Conference on Artificial Intelligence (CORE C)*, 2010.

[57] M. Salido, M. Rodriguez-Molins, and F. Barber. Towards the integration of berth allocation and container stacking problems in martime container terminals. In *The International Conference on Harbor, Maritime & Multimodal Logistics Modelling and Simulation, HMS 2010*, 2010.

[58] M. Salido, O. Sapena, and F. Barber. The container stacking problem: an artificial intelligence planning-based approach. *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling and Simulation*, 2009.

[59] M. Salido, O. Sapena, and F. Barber. What is better: 4 tiers or 5 tiers in the container stacking problem? *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling and Simulation*, 2009.

[60] M. Salido, O. Sapena, M. Rodriguez, and F. Barber. Heuristicas dependientes del dominio en terminales de contenedores. In *PSCS'09 Workshop for Planning, Scheduling and Constraint Satisfaction*, 2009.

[61] M.A. Salido, M. Rodriguez-Molins, and F. Barber. A Decision Support System for Managing Combinatorial Problems in Container Terminals. *Journal Knowledge Based Systems (JCR: 1.308)*, 2010.

[62] M.A. Salido, M. Rodriguez-Molins, and F. Barber. Integrated intelligent techniques for remarshaling and berthing in maritime terminals. *Advanced Engineering Informatics (JCR: 1.73)*, In Press, Corrected Proof:–, 2010.

[63] M.A. Salido, O. Sapena, M. Rodriguez, and F. Barber. A planning-based approach for allocating containers in maritime terminals. In *CAEPIA'09: Thirteenth Spanish Conference for Artificial Intelligence*, 2009.

[64] M.A. Salido, O. Sapena, M. Rodriguez, and F. Barber. A Planning Tool for Minimizing Reshuffles in Container Terminals. In *Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on*, pages 567–571. IEEE, 2009.

[65] O. Sapena and E. Onaindía. Domain independent on-line planning for strips domains. *In proc. IBERAMIA-02, 2527*, pages 825–834, 2002.

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

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

[68] S. Theofanis, M. Boile, and M.M. Golias. Container terminal berth planning: Critical review of research approaches and practical challenges. *Transportation Research Record: Journal of the Transportation Research Board*, 2100:22–28, 2009.

[69] I.F.A. Vis and R. De Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147:1–16, 2003.

[70] Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. MIT. Cent. Space Res., Cambridge, MA, 1971.

[71] P. Zhou, H. Kang, and L. Lin. A dynamic berth allocation model based on stochastic consideration. *Proceedings of the Sixth World Congress on Intelligent Control and Automation (WCICA 2006)*, 2006.