Rodríguez Molins, M.; Salido Gregorio, MA.; Barber Sanchís, F. (2012). Intelligent planning for allocating containers in maritime terminals. Expert Systems with Applications. 39(1):978-989. doi:10.1016/j.eswa.2011.07.098.

Intelligent Planning for Allocating Containers in Maritime Terminals

M. Rodriguez-Molins[a,*], M. A. Salido[a], F. Barber[a]

[a] *Instituto de Automática e Informática Industrial*
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

Maritime container terminals are facilities where cargo containers are transshipped between ships or between ships and land vehicles (tucks or trains). These terminals involve a large number of complex and combinatorial problems. One of them is related to the Container Stacking Problem. A container yard is a type of temporary store where containers await further transport by truck, train or vessel. The main efficiency problem for an individual stack is to ensure easy access to containers at the expected time of transfer.

Stacks are 'last-in, first-out' storage structures where containers are stocked in the order they arrive. But they should be retrieved from the stack in the order (usually different) they should be shipped. This retrieval operation should be efficiently performed, since berthing time of vessels and the terminal operations should be optimized. To do this, cranes can relocate containers in the stacks to minimize the rearrangements required to meet the expected order of demand for containers.

In this paper, we present a domain-dependent heuristically guided planner for obtaining the optimized reshuffling plan, given a stacking state and a container demand. The planner can also be used for finding the best allocation of containers in a yard-bay in order to minimize the number of reshuffles as well as to be used for simulation tasks and obtaining conclusions about possible yard configurations.

Keywords: Planning, Heuristics, Optimizing, Container Stacking Problem

1. Introduction

Maritime container terminals are the most important locations for transshipment and intermodal container transfers (Figure 1). [5] shows how this transshipment market is growing fast (container throughput has increased by 58 per cent over 2000-2004) and needs further studies to analyze it. In order to ensure reliability, e.g. delivery dates or handling times, to the different shipping companies as well as increasing productivity and container throughput from the quayside and landside and vice versa, there are several issues which need optimization. [18, 17] provide an extensive survey about operations at seaport container terminals and methods for their optimization. Moreover, other problems could be faced as for instance planning the routes for liner shipping services to obtain the maximal profit [2]. Another important issue for the success at any container terminal is to forecast container throughput accurately [1]. With this data they could develop better operational strategies and investment plans.

Containers are an ISO standardized metal box and can be stacked on top of each other. Loading and offloading containers on the stack 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.

Figure 2 left shows a container yard. A yard consists of several blocks, and each block consists of 20-30 yard-

*Corresponding author
Email addresses: mrodriguez@dsic.upv.es (M. Rodriguez-Molins), msalido@dsic.upv.es (M. A. Salido), fbarber@dsic.upv.es (F. Barber)

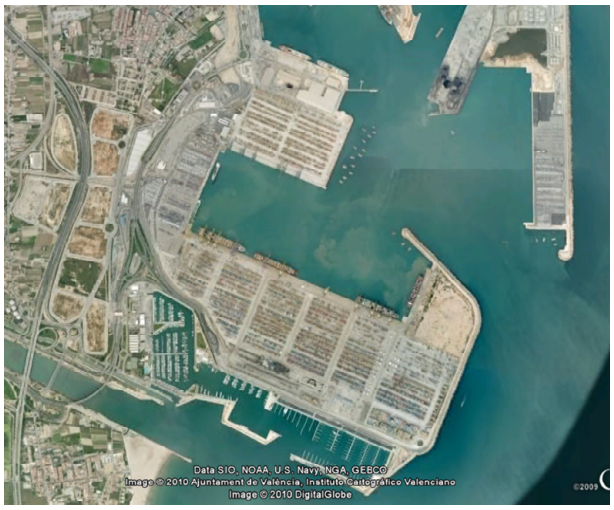


Figure 1: Container Terminal at Valencia

bays [10]. 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 right shows a gantry crane that is able 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 [12], therefore these movements only take place in the same yard-bay.

Figure 2: A container yard (left) and gantry cranes (right) (Photos by Stephen Berend)

When a container arrives at the terminal port, a transfer crane picks it up and stacks it in a yard-bay. During the ship loading operation, a transfer crane picks up the container and transfers it to a truck that delivers it to a quay crane.

In container terminals, the loading operation for export containers is pre-planned by load planners. For load planning, a containership 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. In order to have an efficient load sequence, storage layout of export containers must have a good configuration.

The main focus of this paper 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 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.

In the literature, generally this problem can be seen in two different ways according to when it should be done the optimization:

1. minimizing the number of relocations during the pickup operation.
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.

[11] proposes 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, [9] compares two methods, branch-and-bound algorithm and a heuristic rule based on an estimator, which they minimize the number of relocations during the pickup operation.

In [8], they also propose a methodology to convert the current bay 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 [13] to obtain a plan for remarshalling in automated container terminals.

This paper focuses on this latter issue. But we present a new heuristic with a set of optimization criteria in order to achieve efficiency and take into account constraints that should be considered in real-world problems in the provided solutions.

2. Problem description (The Container Stacking Problem)

The Container Stacking Problem can be viewed as a modification of the *Blocks World* planning domain [19], 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) [3] whose purpose is to express the physical properties of the domain under consideration and it can be graphically represented by means of tools as [4]. A classical AI 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 and the **problem file**, which describes the particular characteristics of each problem. These two files will be described in the following subsections.

2.1. 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, that is, 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 3 the specification 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 3: Formalization of the *stack* operator in *PDDL*.

2.2. 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).

Since the Container Stacking Problem can be formalized with these two files, we can use a general domain independent planner to solve our problems as *Metric FF* [7]. 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 4 shows an example of the obtained plan for a given problem. The performance of this general planner will be analyzed in Section 6, which will be compared with the domain-oriented planner presented in next Sections.

3. 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.

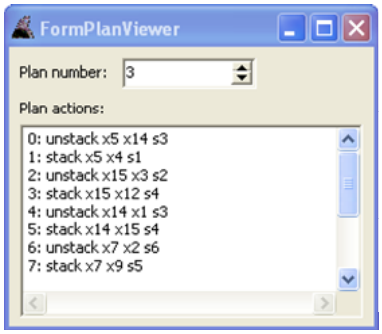


Figure 4: The obtained plan solution to be carried out by the transfer crane.

The proposed planner is built on the basis of a local search domain-independent planner called *Simplanner* [16]. 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 that 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 [6] 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). 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. 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.

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 ← h + Δh;
end
```

4. Optimization criteria for one-bay yards

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

From this heuristic h_1, we have developed some optimization criteria each one of them achieving one of the requirements we could face at Container Terminals [15]. These criteria are centered in the next issues:

1. Reducing distance of the goal containers to the cargo side (OC_{1d}).
2. Increasing the range of the move actions set for the cranes allowing to move a container to 5th tier (OC_{1t}).
3. Applying different ways of balancing within the same bay in order to avoid *sinks* (OC_{1b}).

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

$$h(s) = \alpha \cdot h_1(s) + \beta \cdot h_2(s) \quad (1)$$

being this secondary function a combination of these three criteria described:

$$h_2(s) = OC_{1d} + OC_{1t} + OC_{1b} \quad (2)$$

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 done.

4.1. OC_{1d}: Placing goal containers close to cargo side

Given an initial state, several different layouts can be usually achieved making the same number of reshuffles and some of them can be more interesting than the rest according to other 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.

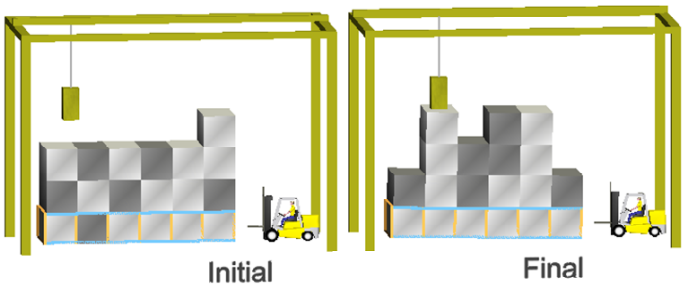


Figure 5: Obtained plan with the initial domain-dependent heuristic.

Following the heuristic function presented in Equation 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 5 and Figure 6. In the first one we want only 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.

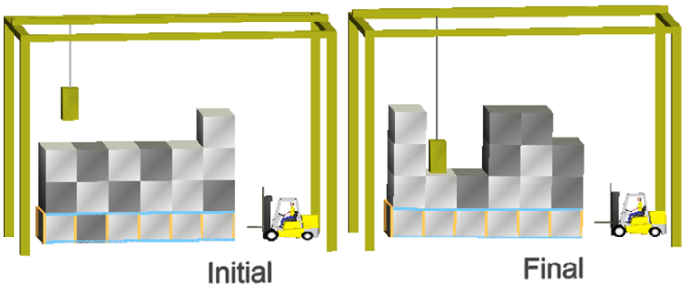


Figure 6: Obtained plan with the distance optimization function.

4.2. 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 which cranes have to be bought depending on how many tiers cranes work. This topic has been considered in [14]. But, another approach is to reach the fifth tier only during the remarshalling process. Thereby, there would be 4 tiers at the beginning and the end keeping the first requirements.

Following this concept, we will use instances of problems $< n, 4 >$ 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.

4.3. 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 7, this criterion avoids the *sinks* in the final layout while all the containers are still in the yard-bay.
   However, when these containers are removed, it might cause that the new layout is unbalanced as it happens in Figure 7(c).

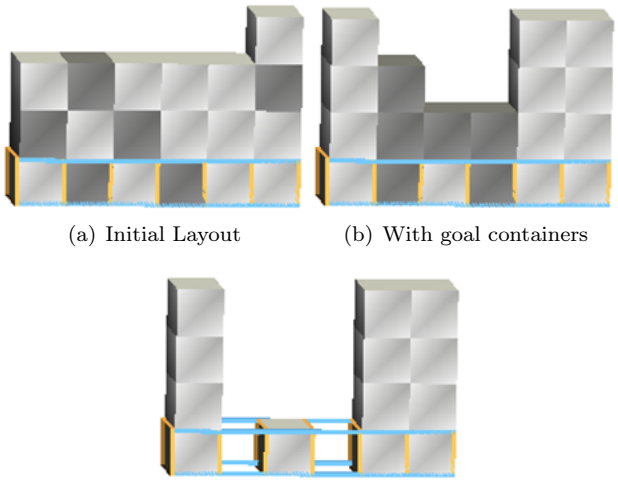
Algorithm 3: Pseudo-code of the domain-dependent heuristic function to allow 5 tiers

```
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 ← h + Δh;
end
```

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]) − 2;
    if Δh > 0 then
        h ← h + Δh;
    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 8 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 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 whether or not there are goal containers on the floor.



(a) Initial Layout (b) With goal containers

(c) Without goal containers

Figure 7: 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

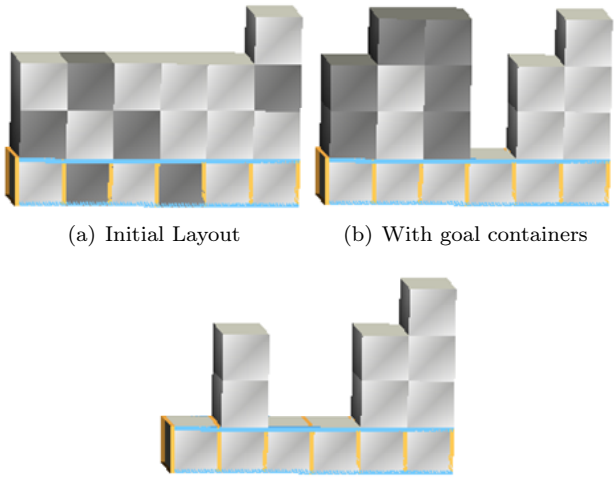
```
Data: b: state of the yard-bay;
Result: MinHeight, heights calculated;
for r ← 1 to numRows(b) do
    MinHeight[r, b] ← Height[r, b];
    // Decrease till the first no goal-container
    while MinHeight[r, b] > 0 ∧
    GoalContainer(MinHeight[r, b], r) ∈ b do
        MinHeight[r, b] ← MinHeight[r,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

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 8(b)). But in this case, nondesirable 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 8(c)).

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 we want to solve the problems seen in the last subsections as we can see it in Figure 9.
   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 contiguous 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 contiguous rows is higher than 2.

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
        if MinHeight[r, s] > 0 then
            Δh ← Δh + NumContainersOn(x);
        end
    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
```



(a) Initial Layout (b) With goal containers

(c) Without goal containers

Figure 8: Effects of using function seen in Algorithm 6

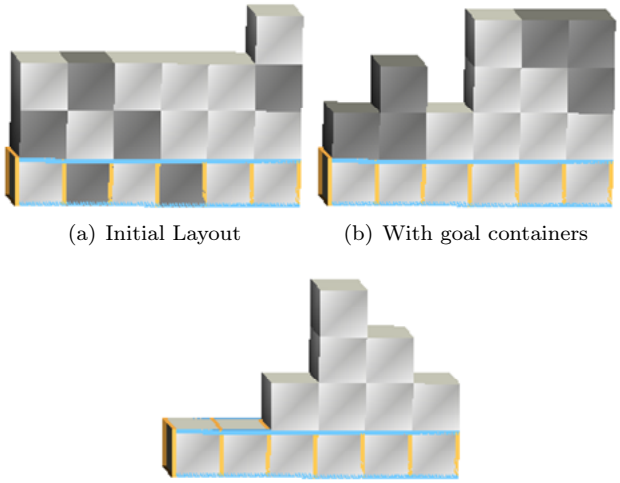
5. Optimization criteria for one block

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 paper, we also have developed two optimization criteria that include new constraints that involve several yard-bays. These constraints are:

- Balancing contiguous yard-bays: rows of adjacent yard-bays must be balanced, 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 10 shows which rows must be get balanced when we consider one yard-bay and Figure 11 left shows an example of non-balanced yard-bays (rows in dotted points).

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
        if MinHeight[r, s] > 0 then
            Δh ← Δh + NumContainersOn(x);
        end
    end
end
h ← h + Δh;
if h < 2 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 + 0.6 × (Δh − 2);
        end
    end
    if h < 2 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 + 0.4 × (Δh − 2);
            end
        end
    end
end
```



(a) Initial Layout (b) With goal containers

(c) Without goal containers

Figure 9: Effects of using function seen in Algorithm 7

- Dangerous containers: two dangerous containers must maintain a minimum security distance. Figure 11 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 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 types of constraints.

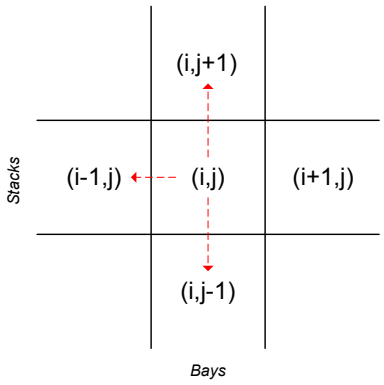


Figure 10: Balancing scheme

These two criteria are added to the heuristic function seen in Equation 1 as h_3 (Equation 3); and Equation 4 shows the exact combination of them. This makes possible to follow a criterion with major priority than the other one.

$$h = \alpha \cdot h_1 + \beta \cdot h_2 + \gamma \cdot h_3 \tag{3}$$

$$h_3 = \delta_1 \cdot OC_{nB} + \delta_2 \cdot OC_{nD} \tag{4}$$

As a consequence of the solving mode followed, depending on the order the yard-bays are resolved may not be possible to achieve a solution. Moreover, as mentioned in Section 4, 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.

5.1. 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. 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

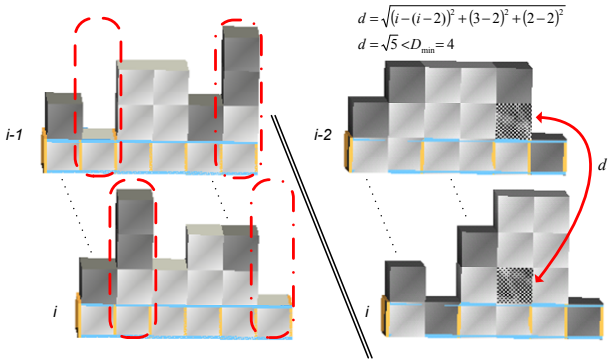


Figure 11: (Left) Non-balanced yard-bays. (Right) Proximity of two dangerous containers.

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 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. This 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.

5.2. 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 enough far 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 as the Euclidean distance, considering each container located in a 3-dimensional space (X,Y,Z)

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 ← h + BalBeforeAfter(b);
// This heuristic will be executed after a partial solution
if h < 1 ∧ NumBay(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 + 0.6 × (Δh − 2);
        end
    end
    if h = 0 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 + 0.4 × (Δh − 2);
            end
        end
    end
end
```

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}.

6. Evaluation

In this section, we evaluate the behavior of the heuristic with the set of optimization criteria presented in this paper. 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 are 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 1 presents the average running time (in milliseconds) to achieve a first solution as well as the best solution found (number of reshuffles) in 10

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;
Result: h: heuristic value of b;
nBay ← NumBay(b);
if nBay > 1 ∧ ∃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);
                if Clear(C_d1) ∈ b then
                    Δh ← Δh + (D_min − dist);
                end
            end
        end
    end
    h ← 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 ← nSinks + 1;
        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 ← nSinks + 1;
            end
        end
    end
end
```

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 ← 1 to numYards(B) do
    nBay ← NumBay(b);
    if nBay > 1 ∧ ∃C_d1 ∈ b then
        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
                    nDang ← nDang + 1;
                end
            end
        end
    end
end
```

seconds for our domain-dependent planner and the average running time (in milliseconds) 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 time for finding a solution and also, this solution 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 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	Time first solution	Best Solution in 10 secs
$< 13, 4 >$	22	3.07	2	3.07
$< 15, 4 >$	3102	4.04	5	3.65
$< 17, 4 >$	4669	5.35	11	4.35
$< 19, 4 >$	6504	6.06	22	4.72
$< 20, 4 >$	22622	7.01	33	5.22
$< 21, 4 >$	13981	6.82	62	5.08

Now we show the effects of using each one of the criteria described in Section 4 separately. In Table 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 finding solution plans that place the selected containers closer to the cargo side of the yard-bay.

Table 2: Average distance obtained by considering distance or not in our domain-dependent heuristic $< n, 4 >$ with 4 tiers.

Instance	***Metric FF***		***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	8.33	4.85
$< 20, 4 >$	12.71	7.01	7.75	5.55
$< 21, 4 >$	12.20	6.82	8.22	5.33

Applying the criterion or function showed in Algorithm 3 we obtain the results appeared in Table 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 greater number of containers, the fewer 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 3: Number of solved problems $< n, 4 >$ with 4 and 5 tiers during the process.

Instance	**4 tiers h_1**	**5 tiers 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 4.3. As we mentioned in this section, since the last function (Algorithm 7) presents the best results after the whole process of remarshalling, we do the comparison in Table 4 among the solutions given by *Metric FF* planner, the initial one h_1 and OC_{1b} (*Both*) in 50 test cases. These results are the average of the best solutions found given a time limit of 1 second for the instances of both $< 15, 4 >$ and $< 17, 4 >$.

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 contiguous 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.

Table 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		**h_1**		**OC_{1b}**	
	(1)	(2)	(1)	(2)	(1)	(2)
Reshuffles	3.72	4.24	3.42	3.72	4, 76	5.04
Sinks	0.62	0.50	0.94	0.66	0	0
Time First	2621	2961	5	9	32	44

From here we realize an evaluation for the criteria presented in Section 5. Table 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 2 blocks of 20 yard-bays. Thus, the results showed in Table 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.

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}).

In this table, 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 18.00 sinks in the block of yard-bay and h_1 generated and average of 29.50 sinks. And the same thing happens for the average number of unfeasible constraints for dangerous containers, *Metric FF* gives us 16.00 and h_1 obtains 7.50.

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 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	3.65	3.38	4.85	4.00	5.65
Sinks	18.00	29.50	0	40.33	0
Non-Safe Dangerous	16.00	7.50	8.00	0	0

7. Conclusions

This paper presents domain-dependent heuristics and a set of optimization criteria for solving the Container Stacking problem by means of planning techniques from Artificial Intelligence. 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 keeping a security distance between dangerous containers, 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 of blocks of yard-bays. 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 vary and determine the minimal number of reshuffles.

This proposed planner with a domain-dependent heuristic allows us obtaining 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 [14] the proposed planner has been applied for obtaining an evaluation of alternative 4 or 5 tiers stacks configuration.

Acknowledgment

This work has been partially supported by the research projects TIN2010-20976-C02-01 (Min. de Ciencia e Innovación, Spain), P19/08 (Min. de Fomento, Spain-FEDER) and the VALi+d Program of the Conselleria d'Educació (Generalitat Valenciana), as well as with the collaboration of the maritime container terminal MSC (Mediterranean Shipping Company S.A.).

References

[1] Chen, S.-H., Chen, J.-N., 2010. Forecasting container throughputs at ports using genetic programming. Expert Systems with Applications 37 (3), 2054 – 2058.
URL http://www.sciencedirect.com/science/article/B6V03-4WNXTWY-M/2/1a5e0fe084ba3ea36303bd280acecc04

[2] Chuang, T.-N., Lin, C.-T., Kung, J.-Y., Lin, M.-D., 2010. Planning the route of container ships: A fuzzy genetic approach. Expert Systems with Applications 37 (4), 2948 – 2956.
URL http://www.sciencedirect.com/science/article/B6V03-4X7YNF2-7/2/19d11092d4e05e1daabb09291c7aa78d

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

[4] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., Vlahavas, I., 2010. A visual programming system for automated problem solving. Expert Systems with Applications 37 (6), 4611 – 4625.
URL http://www.sciencedirect.com/science/article/B6V03-4XX23MG-C/2/3b6b43536b6d90488631a7c780a02df1

[5] Henesey, L., 2006. Overview of Transshipment Operations and Simulation. In: MedTrade conference, Malta, April. pp. 6–7.

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

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

[8] Kim, K., Bae, J., 1998. Re-marshaling export containers in port container terminals. Computers & Industrial Engineering 35 (3-4), 655 – 658, selected Papers from the 22nd ICC and IE Conference.
URL http://www.sciencedirect.com/science/article/B6V27-3WXX91K-2D/2/a56e762628a5a1107f0d067a9dd3af36

[9] Kim, K., Hong, G., 2006. A heuristic rule for relocating blocks. Computers & Operations Research 33 (4), 940–954.

[10] Kim, K., Park, Y., Ryu, K., 2000. Deriving decision rules to locate export containers in container yards. European Journal of Operational Research 124, 89–101.

[11] Kim, K. H., 1997. Evaluation of the number of rehandles in container yards. Computers & Industrial Engineering 32 (4), 701 – 711, new Advances in Analysis of Manufacturing Systems.
URL http://www.sciencedirect.com/science/article/B6V27-3SN7946-2/2/4cf015e963d2d47389862ee31e485b58

[12] Lee, Y., Hsu, N.-Y., 2007. An optimization model for the container pre-marshalling problem. Computers & Operations Research 34 (11), 3295 – 3313.
URL http://www.sciencedirect.com/science/article/B6VC5-4JBGKD8-1/2/2a8c21229ed83e829dd4ef7bdc1fab48

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

[14] Salido, M., Sapena, O., Barber, F., 2009. 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.

[15] Salido, M., Sapena, O., Rodriguez, M., Barber, F., 2009. A planning tool for minimizing reshuffles in containers terminals. ICTAI 2009: 21st International Conference on Tools with Artificial Intelligence.

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

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

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

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