An Artificial Intelligence Planning tool for The Container Stacking Problem

Miguel A. Salido, Oscar Sapena, Federico Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia
Valencia, Spain
{msalido,osapena,fbarber}@dsic.upv.es

Abstract— **Nowadays, there exist a large competition between maritime ports, so that the improvement of customer service became a serious important problem within port container terminals which led to several sub-problems [8]. One of the performance measures is the time spent by vessels in the port quays, so that efficient management of resources in and around a port is essential. A container stack 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. Since stacks are 'last-in, first-out', and the cranes used to relocate containers within the stack are heavily used, the stacks must be maintained in a state that minimizes on-demand relocations. In this paper, we present a planning tool for finding the best configuration of containers in a bay. Thus, given a set of outgoing containers, our planning tool minimizes the number of relocations of containers in order to allocate all selected containers in an appropriate order to avoid further reshuffles. We evaluate the behavior of our tool to analyze the best configuration of yard-bays based on the number of containers, number of selected containers and tiers.**

I. Introduction

Loading and offloading containers on the stack is performed by cranes. In order to access a container which is not at the top of its pile, those above it must be relocated. This reduces the productivity of the cranes.

Maximizing the efficiency of this process leads to several requirements. First, each incoming container should be allocated a place in the stack which should be free and supported at the time of arrival. Second, each outgoing container should be easily accessible, and preferably close to its unloading position, at the time of its departure. In addition, the stability of the stack puts certain limits on, for example, differences in heights in adjacent areas, the placement of empty and 'half' containers and so on.

Since the allocation of positions to containers is currently done more or less manually, this has convinced us that it should be possible to achieve significant improvements of lead times, storage utilization and throughput using improved techniques of the type indicated.

Figure 1 shows a container yard. A yard consists of several blocks, and each block consists of 20-30 yard-bays [6]. Each yard-bay contains several (usually 6) rows. When an outside truck delivers an outbound container to a yard, 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.

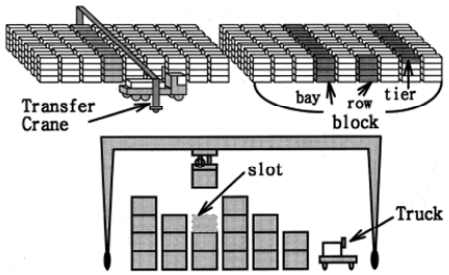

Fig. 1. A container yard [6].

In container terminals, the loading operation for export containers is carefully pre-planned by load planners. For load planning, a containership agent usually transfers a load profile (an outline of a load plan) to terminal operating company several days before a ship's arrival. The load profile specifies only the container group, which is identified by container type (full or empty), port of destination, and size to be stowed in each particular ship cell. Since a ship cell can be filled with any container from its assigned group, the handling effort in the marshalling yard can be made easier by optimally sequencing export containers in the yard for the loading operation. In sequencing the containers, load planners usually pursue two objectives:

1) Minimizing the handling effort of quay cranes and yard equipment.
2) Ensuring the vessel's stability.

The output of this decision-making is called the "load sequence list". 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 optimally reallocating 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.

This work has been partially supported by the research projects TIN2007-29666-E and TIN2007-67943-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER).

978-1-4244-2728-4/09/$25.00 ©2009 IEEE

Given a layout, the user selects the set of containers that will be moved to the vessel. Our tool is able to organize the layout in order to allocate these containers at the top of the stacks in order to minimize the number of relocations. Thus a solution of our problem is a layout where all outgoing containers can be available without carrying out any reshuffle. Furthermore, due to harbor operator requirements, we are interesting on comparing the number of reshuffles in yard-bays with 4 tiers against yard-bays with 5 tiers.

II. The problem modelled as an Artificial Intelligence planning problem

A classical AI planning problem can 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.

The container stacking problem is a slight modification of the *Blocks World* planning domain [7], 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 positioning of the towers on the table is irrelevant. 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.

This problem is closed to the container stacking problem, but there are some important differences:

- The number of towers is limited in the container stacking problem: a yard-bay contains usually 6 rows, so it is necessary to include an additional constraint to limit the number of towers on the table to 6.
- The height of a tower is also limited. In this paper we analyze the effect of limiting the number of levels in a tower on the number of required relocations to reach the goal configuration.
- 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) [4]. Following this standard, a planning task is defined by means of two text files: the domain file, which contains the common features for all problems of this type, and the problem file, which describes the particular characteristics of each problem. The contents of both files are described in the following subsections with more detail.

A. The container stacking domain

The main elements in a domain specification are (1) the types of objects we need to handle, (2) the types of propositions we use to describe the world and (3) the actions we can perform to modify the state of the world. In the container stacking domain we have the following elements:

1) *Object types*. In this domain we only need to define two object types: *containers* and *rows*, where the rows represent the areas in a yard-bay in which a tower of containers can be built.
2) *Propositions*. We need to define the following types of propositions:
   - `on ?x - container ?y - (either row container)`
     This predicate indicates that the container `?x` is on `?y`, which can be another container or, directly, the floor of a row (stack).
   - `at ?x - container ?r - row`
     This indicates that the container `?x` is in the tower built on the row `?r`.
   - `clear ?x - (either row container)`
     This predicate states that `?x`, which can be a row or a container, is clear, that is, there are no containers stacked on it.
   - `crane-empty`
     This indicates that the crane used to move the containers is not holding any container.
   - `holding ?x - container`
     This states that the crane is holding the container `?x`.
   - `goal-container ?x - container` and `ready ?x - container`
     These predicates are 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.
   - `height ?s - row` and `num-moves`
     These are 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.
3) *Actions*. In this domain there are four different actions to move the containers from a row to another:
   - `pick (?x - container ?r - row)`
     With this action the crane picks the container `?x` which is in the floor of row `?r`.
   - `put (?x - container ?r - row)`
     The crane puts the container `?x`, which is holding, in the floor of row `?r`.
   - `unstack (?x ?y - container ?r - row)`
     With this action the crane unstacks the container `?x`, which is in row `?r`, from the container `?y`.

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

Finally, we have defined two fictitious actions that allow to check whether a given (goal) container is ready, that is, it is in a valid position:

- The container is clear, or
- The container is under another (goal) container which is in a valid position.

B. A container stacking problem

A container stacking problem file contains the elements which are specific to the particular problem. These elements are:

- *Objects*: the rows available in the yard-bay 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).

III. Domain-independent planning for solving the container stacking problem

Since the container stacking problem can be formalized in *PDDL* format, as we have shown in the previous section, we can use a general planner to solve our problem instances. Currently we can found several general planners which work well in many different domains, such as *LPG-TD* [3], *MIPS-XXL* [2] and *SGPlan* [1]. However, and due to the high complexity of the domain we are handling, these planners are not able to find good plan solutions efficiently. *LPG-TD*, for example, spends too much time in the preprocessing stages, so it takes a long time to provide a solution. On the contrary, *MIPS-XXL* and *SGPlan* can compute a solution rapidly, but the quality of the obtained solution is not good enough, including some additional relocation movements to achieve the goal configuration.

In order to solve this problem efficiently, we have developed a new general planning algorithm with several interesting properties for the container stacking problem:

- It is an anytime planning algorithm [9]. 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.
- The planner is complete, so it will always find a solution if exists.
- The planner is optimal. It guarantees finding the optimal plan if there is time enough for computation.

The planning approach is a combination of an *Enforced Hill-Climbing* [5], which allows to find fast solutions, with a standard A search, which guarantees finding the optimal plan, that is, the plan that minimizes the number of reshuffles.

Algorithm 1 shows the planner's main loop. The planner stores a list of unexplored states, *open_nodes*, which initially only contains the initial state I. At each iteration, the planner selects from that list the next state to visit according to a linear combination of two functions: $g(s)$, which is the cost to reach s from the initial state and $h(s)$, which is the estimated cost to reach the goals from s. This heuristic is computed as the number of actions of a relaxed plan which ignores the delete effects of the operators [5]. Then, a hill-climbing search is started from the selected state.

Data: $\langle A, I, G \rangle$
$open\_nodes = \{I\}$;
while $open\_nodes \neq \emptyset$ **do**
  $s = \arg\min g(s') + h(s'), \forall s' \in open\_nodes$;
  if $h(s) = 0$ **then** *solutionFound*(s);
  hillClimbing($s, \langle A, I, G \rangle$, $open\_nodes$);
end

Algorithm 1: Pseudocode for planner's main loop

Hill-climbing is a local search that always moves to the best child node according to its heuristic value, see Algorithm 2. The main bottleneck while solving the container stacking problem is the large number of local minima (or plateaux) found. A plateau appear when no child node improves the heuristic value of the current node and, in this case, a special search, shown in Algorithm 3, is started to find an exit node from which to continue with the local search.

Data: s, $\langle A, I, G \rangle$, $open\_nodes$
$cont = true$;
repeat
  $children = \{s' / \exists a_i \in A : s' = result(a_i, s)\}$;
  if $best\_child \neq \emptyset$ **then**
    $best\_child = \arg\min h(s'), \forall s' \in children$;
    $open\_nodes = open\_nodes \cup children \setminus best\_child$;
    if $h(best\_child) \geq h(s)$ **then**
      $best\_child$ = *SGAsearch*(s, $\langle A, I, G \rangle$, $open\_nodes$);
      if $best\_child$ = *none* **then** $cont = false$
    $s = best\_child$;
    if $h(s) = 0$ **then** *solutionFound*(s);
  else
    $cont = false$;
  end
until $cont = false$;

Algorithm 2: Hill-climbing algorithm

Data: s, $\langle A, I, G \rangle$, $open\_nodes$
Result: Plateau exit node
$f(n) = 0.1 * g(n) + 0.9 * sgh(n)$;
$plateau\_nodes = \{s\}$;
while $plateau\_nodes \neq \emptyset$ **do**
  $s_{next} = \arg\min f(s'), \forall s' \in plateau\_nodes$;
  $plateau\_nodes = plateau\_nodes \setminus \{s_{next}\}$;
  if $sgh(s_{next}) = 0$ **then** $f(n) = 0.4 * g(n) + 0.6 * h(n)$;
  $children = \{s' / \exists a_i \in A : s' = result(a_i, s_{next})\}$;
  forall $s' \in children$ **do**
    if $h(s') < h(s)$ **then return** s';
    $plateau\_nodes = plateau\_nodes \cup \{s'\}$;
  end
end
$open\_nodes = open\_nodes \cup leafNodesFrom(s)$;
return *none*;

Algorithm 3: Plateau search to escape from local minima

The plateau search is a best-first-search guided by a linear combination of $g(s)$ and a new heuristic function, $sgh(s)$, which only takes into account the most costly goal (instead of all top level goals as $h(s)$ does). The $sgh(s)$ is much more informative than $h(s)$ inside a plateau, so the effort to find an exit node is reduced.

IV. Evaluation

To analyze out tool, we have evaluated the minimum number of reshuffles needed to allocate all selected containers at the top of the stacks or under another selected containers in such a way that no reshuffles is needed to load outgoing containers.

To evaluate the behavior of our tool, the experiments were performed on random instances. A random instance is characterized by the tuple $< n, s >$, where n is the number of containers and s is the number of selected containers. Each instance is a random configuration of all containers distributed along the six stack with 4 or 5 tiers. We evaluated 100 test cases for each type of problem.

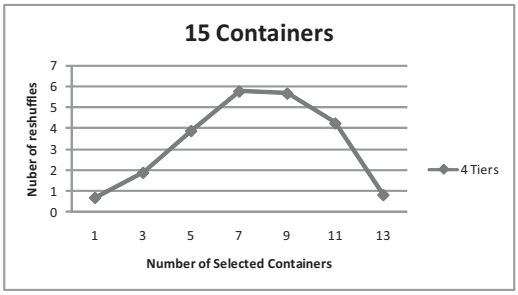

Fig. 2. Number of reshuffles for problems $< 15, s >$ with 4 tiers.

In Figure 2, we evaluated the number of reshuffles needed for problems $< 15, s >$ with 4 tiers. Thus, we fixed the number of containers to 15 and we increased the number of selected containers s from 1 to 13. It can be observed that as the number of selected containers increased, the number of reshuffles increased until a threshold in which the number of reshuffles decreases due to the fact that the number of selected containers is closer to the number of containers. In this configuration, the upper bound is reached when the number of selected containers is 7. We have also evaluated this type of problems ($< 15, 7 >$) with 5 tiers, and the number of reshuffles decreased from 5.7 (with 4 tiers) to 5.1 (with 5 tiers).

In Figure 3, we evaluated the number of reshuffles needed for problems $< n, 4 >$ with 5 tiers. To this end, we fixed the number of selected containers to 4 and we increased the number of containers n from 11 to 23. The figure shows that as the number of selected containers increases, the number of reshuffles also increased.

By analyzing both figures, it can be observed that for problems $< 15, 4 >$, the figure 2 has an approximate value of 3 meanwhile the figure 3 has a value of 2.6. The difference is due to the fact that yard-bays with 5 tiers generates less reshuffles than yard-bays with 4 tiers on this configuration.

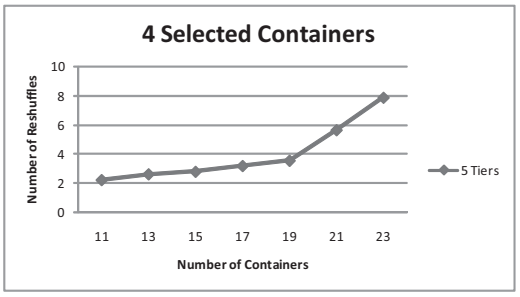

Fig. 3. Number of reshuffles for problems $< n, 4 >$ with 5 tiers.

V. Conclusions and Further Works

This paper presents the modelling of the container stacking problem form the Artificial Intelligence point of view. We have developed a domain-independent planning tool for finding an appropriate configuration of containers in a 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 or under another selected containers in such a way that no reshuffles is needed to load these outgoing containers. We evaluate the behavior of our tool to analyze the best configuration of yard-bays based on the number of containers, number of selected containers and tiers. In further works, we will focus our attention in the development of domain-dependent planning heuristic to include new hard and soft constraints for solving this problem.

References

[1] Y. Chen, C.W. Hsu, and B.W. Wah, 'SGPlan: Subgoal partitioning and resolution in planning', *IPC-4 Booklet (ICAPS)*, (2004).

[2] S. Edelkamp, 'Taming numbers and durations in the model checking integrated planning system', *Journal of Artificial Intelligence Research (JAIR)*, **20**, 195–238, (2003).

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

[4] 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).

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

[6] Kap Hwan Kim, Young Man Park, and Kwang-Ryul Ryu, 'Deriving decision rules to locate export containers in container yards', *European Journal of Operational Research*, **124**, 89–101, (2000).

[7] J. Slaney and S. Thibaux, 'Blocks world revisited', *Artificial Intelligence*, **125**, 119–153, (2001).

[8] 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).

[9] S. Zilberstein and S.J. Russell, 'Optimal composition of real-time systems', *Artificial Intelligence*, **82**(1-2), 181–213, (1996).