THE CONTAINER STACKING PROBLEM: AN ARTIFICIAL INTELLIGENCE PLANNING-BASED APPROACH

Miguel A. Salido[(a)], Oscar Sapena[(b)], Federico Barber[(c)]

[(a)(b)(c)]Instituto de Automática e Informática Industrial, Universidad Politécnica de Valencia, Spain

[(a)]msalido@dsic.upv.es, [(b)]osapena@dsic.upv.es, [(c)]fbarber@dsic.upv.es

ABSTRACT

Large container ports around the world are major hubs in the global cargo transport system. 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. In this paper, we propose 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. Furthermore, we compare the number of reshuffles in yard-bays with 4 tiers against yard-bays with 5 tiers. The obtained results recommend the use of stacks with 5 tiers in high loaded yard-bays, due to the fact that the number of reshuffles is reduced.

Keywords: container-stacking, artificial intelligence, planning

1. 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 (Kim, Park and Ryu 2000). 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.

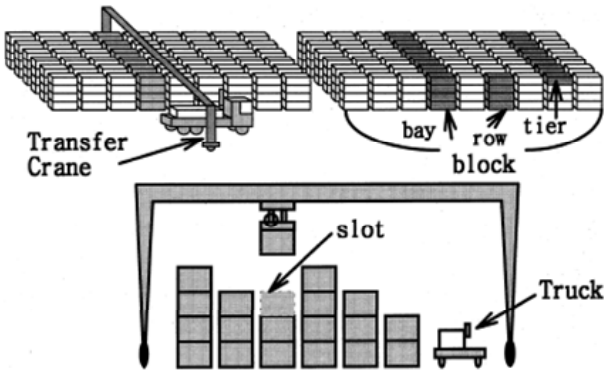

Figure 1: A container yard (Kim, Park and Ryu 2000)

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:

- Minimizing the handling effort of quay cranes and yard equipment.
- 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.

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.

2. THE PROBLEM MODELLED AS AN ARTIFICIAL INTELLIGENCE PLANNING PROBLEM

A classical AI planning problem can defined by a tuple <*A*, *I*, *G*>, 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 (Slaney and Thiébaux 2001), 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* (Ghallab et al. 1998). 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.

2.1. 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: *object types*, *propositions* and *actions*.

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.

Types of propositions. We need to define the following types of propositions:

- *on ?x - container ?y - (either row container)*
  This predicate indicates that 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*
  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*
  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.

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 - container ?y - container ?r - row)*
  With this action the crane unstacks the container *?x*, which is in row *?r*, from the container *?y*.
- *stack (?x - container ?y - container ?r - row)*
  The crane stacks the container *?x*, which is currently holding, on container *?y* in the row *?r*.

As an example, we show in Figure 2 the specification of the stack operator in *PDDL* format. 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: Formalization of the *stack* operator in *PDDL*.

Finally, we have defined two fictitious actions that allow checking 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.

2.2. 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).

3. 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* (Gerevini, Saetti and Serina 2003), *MIPS-XXL* (Edelkamp 2003) and *SGPlan* (Chen, Hsu and Wah 2004). 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 (Zilberstein and Russell 1996). 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 main bottleneck while solving the container stacking problem is the large number of local minima (or plateaux) found during the search. We have developed a new search method which combines the use of two heuristic functions. This feature allows the planner to escape from a local minimum very efficiently.

The planning approach is a combination of an *Enforced Hill-Climbing* (Hoffman and Nebel 2001), 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.

The plan is returned by the planner as a totally ordered sequence of actions. Figure 3 shows the obtained plan for a given problem. This plan shows the movements the transfer crane must carry out to achieve our objective.

Figure 3: Plan to be carried out by the transfer crane.

4. EVALUATION

To evaluate our tool, we have analyzed two configurations of yards: with 4 tiers and with 5 tiers. 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. Thus, 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 stacks with 4 or 5 tiers. We evaluated 100 test cases for each type of problem.

In Figure 4 we evaluated the number of reshuffles needed for problems <*n*, 4>. Thus, we fixed the number of selected containers to 4 and we increased the number of containers *n* from 11 to 23. It can be observed that as the number of containers increased, the number of reshuffles increased. For small number of containers (low values of *n*) there is no difference between 4 tiers and 5 tiers. This is due to the fact that it is not needed the use of the higher stacks to achieve a solution because there exist many combinations to achieve a solution. However, the number of reshuffles with 5 tiers was lower than the number of reshuffles with 4 tiers for higher number of containers. Due to the fact that we consider yard-bays of 6 stacks, for problems with 4 tiers the maximum number containers is bounded to 24. In this case instances <23, 4> for problems with 4 tiers generally has no solution. Thus, we can conclude that for low loaded yard-bays (< 15 containers) there is not different between 4 tiers and 5 tiers, meanwhile for high loaded yard-bays 5 tiers is more appropriate for minimizing the number of reshuffles.

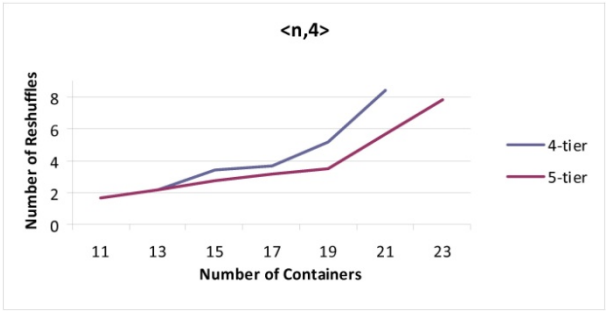

Figure 4: Number of reshuffles for problems <*n*, 4>.

In Figure 5 we evaluated the number of reshuffles needed for problems <19, *s*>. To this end, we fixed the number of containers to 19 and we increased the number of selected containers *s* from 2 to 6. The figure shows that as the number of selected containers increases, the number of reshuffles also increased. It can be also observed that the number of reshuffles for tiers 5 was lower than the number of reshuffles for tiers 4 in all cases. Furthermore, as the number of selected containers increases, the difference of reshuffles with 4 and 5 tiers became higher, so we can conclude that configurations of yards with 5 tiers is more appropriate to minimize the number of reshuffles with this configuration.

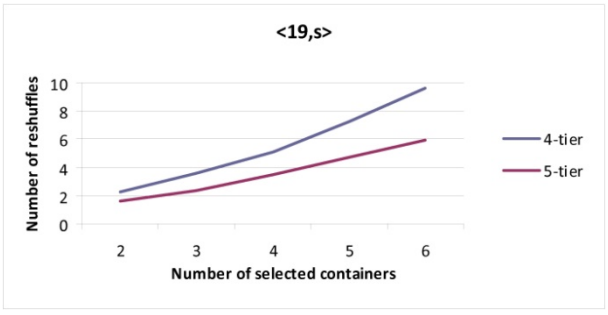

Figure 5: Number of reshuffles for problems <19, *s*>.

5. CONCLUSIONS AND FURTHER WORKS

This paper presents the modeling 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.

Furthermore, we compare the number of reshuffles in yard-bays with 4 tiers against yard-bays with 5 tiers. The obtained results recommend the use of stack with 5 tiers for high loaded yard-bays, due to the fact that the number of reshuffles is reduced for outgoing containers.

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.

ACKNOWLEDGMENTS

This work has been partially supported by the research projects TIN2007-67943-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER) , P19/08 (Min. de Fomento, Spain-FEDER) and by the Technical University of Valencia.

REFERENCES

Chen, Y., Hsu, C.W., Wah, B.W., 2004. SGPlan: Subgoal Partitioning and Resolution in Planning. *IPC-4 Booklet (ICAPS)*.

Edelkamp, S., 2003. Taming Numbers and Durations in the Model Checking Integrated Planning System. *Journal of Artificial Intelligence Research (JAIR)*, 20, 195–238.

Gerevini, A., Saetti, A., Serina, I., 2003. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *Journal of Artificial Intelligence Research (JAIR)*, 20, 239–290.

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*.

Hoffman, J., Nebel, B., 2001. The FF Planning System: Fast Planning Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14, 253–302.

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

Slaney, J., Thiébaux, S., 2001. Blocks World revisited. *Artificial Intelligence*, 125, 119–153.

Zilberstein, S., Russell, S.J., 1996. Optimal Composition of Real-time Systems. *Artificial Intelligence*, 82(1-2), 181–213.

AUTHORS BIOGRAPHY

Miguel A. Salido is an associate professor in Computer Science at the Technical University of Valencia, Spain. Most of his research is focused on techniques for constraint satisfaction techniques and railway scheduling problems. He is the recipient of some national and international awards. He is author of more than 70 papers published on international journals and conferences. He is editor of some books and guess editor of some international journals. He is member of several Scientific and Organizing Committees.

Oscar Sapena is an associate professor in Computer Science at the Technical University of Valencia, Spain. His research is focused on Planning form the Artificial Intelligence point of view. He is author of more than 20 papers published on international journals and conferences. He is member of several Scientific and Organizing Committees.

Federico Barber is Full Professor with the Department of Computer Science, where he leads a research team in artificial intelligence. He has worked on the development of temporal reasoning systems, Constraint Satisfaction Problems, planning and scheduling. He is the author of about 90 research articles which have been published in several journals and conferences. His research has produced several tools for solving real-world optimization combinatory problems. He has participated in and led several national and European research projects related to these areas. He is currently president of the Spanish Association for Artificial Intelligence, and member of several scientific associations.