An Investigation of IDA* Algorithms for the Container Relocation Problem

Huidong Zhang[1], Songshan Guo[1], Wenbin Zhu[2,⋆],
Andrew Lim[3], and Brenda Cheang[3]

[1] Department of Computer Science, School of Information Science and Technology
Sun Yat-Sen University, Guangzhou, Guangdong, PR China (510275)
zwdant@gmail.com, issgssh@mail.sysu.edu.cn
[2] Department of Computer Science and Engineering, School of Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong S.A.R.
i@zhuwb.com
[3] Department of Management Sciences, College of Business
City University of Hong Kong, Tat Chee Ave, Kowloon Tong, Hong Kong S.A.R.
{lim.andrew,brendac}@cityu.edu.hk

Abstract. The container relocation problem, where containers that are stored in bays are retrieved in a fixed sequence, is a crucial port operation. Existing approaches using branch and bound algorithms are only able to optimally solve small cases in a practical time frame. In this paper, we investigate iterative deepening A* algorithms (rather than branch and bound) using new lower bound measures and heuristics, and show that this approach is able to solve much larger instances of the problem in a time frame that is suitable for practical application.

Keywords: Container Relocation Problem, IDA*, Heuristics.

1 Introduction

The retrieval of containers out of storage and onto transport vehicles is a common and crucial port operation. The containers in a storage bay must be retrieved one by one in a fixed sequence that is pre-determined by various constraints (e.g., maintenance of vessel balance, safety issues, etc.). The problem arises when the next container to be retrieved is not at the top of its stack, since all other containers above it must then be first relocated onto other stacks within the bay. The relocation of a container is an expensive operation that essentially dominates all other aspects of the problem, and therefore it is important that the retrieval plan minimizes the number of such relocations.

There are two versions of this *Container Relocation Problem* in existing literature: the *restricted* variant only allows the relocation of containers that are above the target container, while the *unrestricted* variant allows the relocation of

⋆ Corresponding Author.

N. García-Pedrajas et al. (Eds.): IEA/AIE 2010, Part I, LNAI 6096, pp. 31–40, 2010.

any container. Previous research has largely concentrated on only the restricted variant, which is also the focus of this study. We propose two new lower bound measures that are superior to the existing lower bound the restricted variant. We also investigate the effects of some greedy heuristics for the purpose of rapid updating of best known solutions during interative depeening A* search. Our research shows that using IDA* for the Container Relocation Problem is far superior to the standard branch and bound approach proposed in existing literature. In particular, we show that the restricted variant can be well solved for instances of practical size using IDA*.

2 Problem Description

In the container yard of a port, containers are stored in *blocks* (see Figure 1); each block consists of multiple *bays*; each bay consists of multiple *stacks* of containers; and each stack consists of multiple *tiers*. Container blocks typically comprise up to 20 bays, with each bay having maximum capacities between 2-10 stacks and 3-7 tiers.

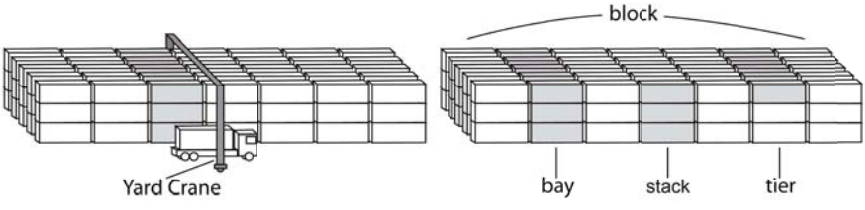


Fig. 1. Blocks in a Container Yard

Containers are retrieved from the bays using yard cranes, loaded onto AGVs, and transported onto quay cranes that finally load them onto vessels. The loading sequence of the containers by the quay cranes seeks to minimize the berth time of the vessel while satisfying various loading constraints (e.g., vessel balance and safety issues); the quay crane loading schedule determines the pickup sequence of the containers from the yards.

This study focuses on the yard crane scheduling problem, where the goal is to produce an operational plan for the retrieval of all containers in a given pickup sequence that minimizes the total time spent. Figure 2 illustrates an instance of the problem, where the numbers represent the pickup sequence of the containers. If the container that is to be retrieved next in the pickup sequence (called the *target* container) is on top of its stack, then it is simply retrieved. However, if this is not the case then all containers on top of the target container must first be relocated onto other stacks in the bay. The nature of yard crane operations is such that minimizing the total time taken by an operational plan is functionally equivalent to minimizing total number of *relocations* ([1,2,3,4,5]).

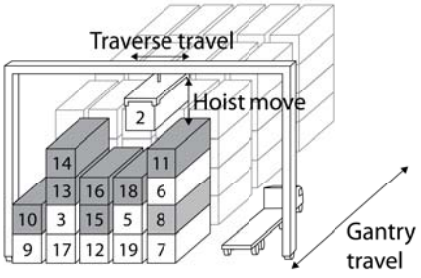


Fig. 2. Yard Crane Operations

3 Existing Approaches

Kim and Hong [4] described a standard branch and bound algorithm for the restricted variant that can optimally solve instances of up to 5 stacks by 5 tiers within 4 minutes, but is too slow for practical use for larger instances. A heuristic rule for generating solutions based on the expected number of relocations for a random operational plan was proposed as an alternative; this heuristic is able to find solutions that are within 10% of optimal for instances of up to 5×5 with running time of up to 2 seconds. While the unrestricted variant was mentioned, no solution was suggested for it.

Kim and Hong's work [4] was extended by Aydin [6], which proposed alternative branching rules for the standard branch and bound algorithm for the restricted variant while using the same lower bound. In combination with two greedy heuristics, the branch and bound algorithm was found to perform well for instances of up to 7 stacks by 7 tiers with about 37 containers, although the test cases had relatively low load factors ($55\% - 75\%$). A variant of the container relocation problem where traverse travel is modeled as a variable cost proportional to the number of stacks traveled was also considered (which is beyond the scope of this study).

Existing work on the storage of inbound containers so as to minimise relocations during retrieval include [7] and [2], which both proposed formulae for estimating the expected number of relocations. Kim et al. [3] developed a mathematical model and used a dynamic programming technique for this purpose, producing a decision tree for use during the storage process.

Avriel et al. [8] proposed a 0-1 integer-programming model and a heuristic method to solve a similar (but not identical) relocation problem of stowing containers onto a vessel.

4 Notation and Terminology

The following notation will be employed in this paper.

L	: A bay layout
N	: Number of containers in the initial bay layout
S	: Maximum number of stacks in the bay
T	: Maximum number of tiers in the bay
$F(L)$	: The minimum number of relocations required for layout L
$c_{s,t}$	: The retrieval order of the container on tier t of stack s
$\bar{s}$	: The stack containing the target
$\bar{t}$	: The tier containing the target; the target container is therefore $c_{\bar{s},\bar{t}}$
$smallest(s)$	: The smallest retrieval order in stack s, i.e., $smallest(s) = \min c_{s,t}$.

By convention, we order the stacks in ascending order $1..S$ from left to right, and the tiers $1..T$ from bottom to top.

5 Iterative Deepening A*

For a given bay layout, if the target container is at the top of its stack, then it can be immediately retrieved, resulting in a smaller bay layout with the same cost in terms of number of required relocations. Successive target containers can be retrieved as long as they are on top of their respective stacks at the time of retrieval, until the minimum equivalent layout is reached where there are either no more containers or there exists other container(s) on top of the target container. Hence, all layouts encountered in the search can be replaced by its minimum equivalent layout.

Every node n in the search tree corresponds to a minimum layout; every path from the root node to a leaf node corresponds to a solution to the initial layout. Branching occurs when relocation takes place. If the bay has S stacks, then every node in the search tree will have at most $S - 1$ children. Figure 3 illustrates the nodes in the first two levels of a search tree of the restricted variant.

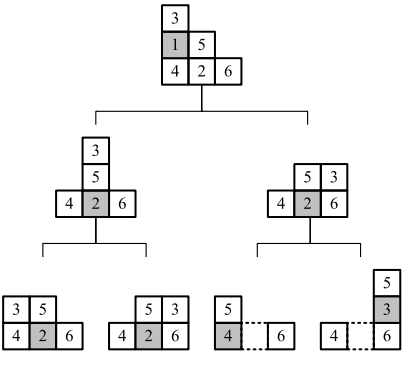


Fig. 3. First 2 Levels of a Search Tree for the Restricted Variant

The approach proposed in this study is an iterative deepening A* (IDA*) algorithm. In the first iteration, a strictly optimistic (i.e., non-overestimating) lower

bound cost threshold t is estimated for the initial problem, which guarantees the permissibility of the solution. A depth-first branch and bound is then performed. Each node n in the search tree has a corresponding cost $f(n) = g(n) + h(n)$, where $g(n)$ is the number of relocations already made to arrive at that node (known as *confirmed* relocations) and $h(n)$ is a strictly optimistic estimation of the minimum number of relocations required to solve the problem from that node (known as *identified* relocations). Over the course of the search, the best lower bound $lb(n)$ for each node n is updated as all its children n' are evaluated, where $lb(n) = \max lb(n') + 1$.

One iteration of IDA* ends when the branch and bound has determined that all remaining nodes n have $f(n) > t$. This process is repeated with the threshold t incremented by 1, or set to the best lower bound found for the initial layout in the previous iteration, whichever is greater. The algorithm is completed when a solution is found with cost equal to t.

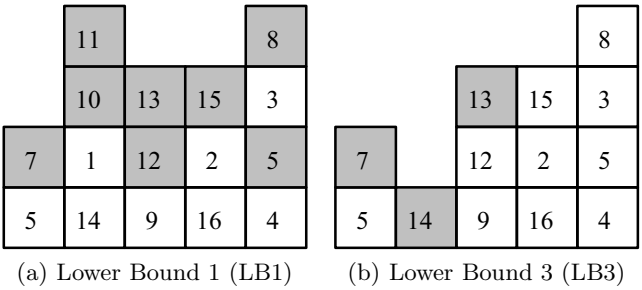
5.1 Lower Bounds

Recall that the estimated cost $f(n)$ of a particular layout is the sum of its confirmed relocations $g(n)$ and its identified relocations $h(n)$. In order for the solution to be admissible, $h(n)$ must never overestimate the number of relocations required. We examine three admissible lower bound measures that can be used as $h(n)$.

Lower Bound 1 (LB1): If a container is situated above another container with smaller retrieval order, then it must be relocated at least once. LB1 counts the number of such containers, i.e.,

$$LB1(L) = |\{c_{s,t} \mid \exists_{t',t'<t} \text{ such that } c_{s,t'} < c_{s,t}\}| \tag{1}$$

All the shaded containers in Figure 4(a) must be relocated at least once. This is the lower bound described by Kim and Hong [4].



(a) Lower Bound 1 (LB1) (b) Lower Bound 3 (LB3)

Fig. 4. Lower Bounds LB1 and LB3

Lower Bound 2 (LB2): Observe in Figure 4(a) that there are three possible destinations to relocate container 11: on stacks 1, 3 and 4 (the rightmost stack is

full and cannot hold any more containers). For all of the three possible choices, there will be a container with a smaller retrieval order below container 11 Hence, container 11 has to be relocated at least one more time. This inspection can be performed on all containers above the target (container 1), and the number of such cases is added to LB1, i.e.,

$$LB2(L) = LB1(L) + |\{c_{\bar{s},t} \mid t > \bar{t} \text{ and } \forall_{s' \neq \bar{s}}, \exists_{t'} \text{ such that } c_{s',t'} < c_{\bar{s},t}\}| \quad (2)$$

Lower Bound 3 (LB3): Given a layout L and a set of containers S, let $L - S$ denote the resultant layout when all containers in S are removed from L. It is apparent that for any layout $L' = L - S$, $F(L') \leq F(L)$. This observation allows an extension of LB2 to all containers (rather than only containers in stack $\bar{s}$), as illustrated by the following example.

Let L be the bay layout given in Figure 4(a). Consider container 15 on stack 4 in L; the container in stack 4 with smallest retrieval order is container 2, which will therefore be the target container when stack 4 is next considered. Suppose all containers with a smaller retrieval order than 2 is removed (container 1 in this case) along with all containers above them (containers 10 and 11 in this example); this results in a smaller layout $L' = L - \{c_{2,2}, c_{2,3}, c_{2,4}\}$ as depicted in Figure 4(b). We can then use the same analysis as LB2 for container 15 in L', i.e., for all of the possible destination stacks for its relocation, there will be a container with a smaller retrieval order than 15. Hence, container 15 must be relocated an additional time.

$$L'_{s,t} = L - \{c_{s',t'} \mid \exists_{t'',t'' \leq t'} \text{ such that } c_{s',t''} < smallest(s)\} \quad (3)$$

$$LB3(L) = LB1(L) + \sum_{s,t}(LB2(L'_{s,t}) - LB1(L'_{s,t})) \quad (4)$$

5.2 Probe Heuristics

The deepest nodes explored in each iteration of IDA* can be viewed as forming a frontier. Subsequent iterations push the frontier further and further towards the leaf of the search tree and stops when the first leaf node is reached.

IDA* requires the computation of $f(n')$ for all children n' of a node n before we can determine if n is on the frontier of the current iteration. It could potentially improve the effectiveness of the search if we could make good use of this information. If a child n' of a node on the frontier seems promising, we could invest the time to complete the partial solution represented by n' using a heuristic, which may turn out to be superior to the best known solution. In particular, we will apply a *heuristic probe* to n' if $f(n') \leq (lb + b)/2$, where lb is the lower bound of the root node found in previous iteration, and b is the cost of the best known solution found so far. The best known solution is updated if the heuristic finds a better solution, thus narrowing the search window.

This subsection describes the heuristics used to generate complete solutions during probing. These heuristics are used when the target container is not on the top of its stack, and provide criteria for moving a container from the top of

one stack to another. We repeat the process until all containers are retrieved. Clearly, the height of destination stack must be less than T (we call such stacks *incomplete stacks*) and the destination stacks must differ from the source stack. In the following discussion, we implicitly assume that only incomplete stacks (other than the source stack) are considered when selecting the destination stack.

We examine four such heuristics.

PR1: The stack with minimum height; if there are multiple eligible stacks, then the leftmost eligible stack (with the smallest number) is selected.

PR1 is a simple heuristic that selects the shortest stack on the premise that it minimizes the possibility that the relocated container will be placed above a container with smaller retrieval order.

PR2: The stack with highest $smallest(s)$ (refer to Section 4).

PR2 is motivated by the observation that if a container c is relocated to a stack with $smallest(s)$ greater than its retrieval order, then container c need not be relocated again in the future. By picking the stack with the highest $smallest(s)$ value, we maximise the probability of this occurence.

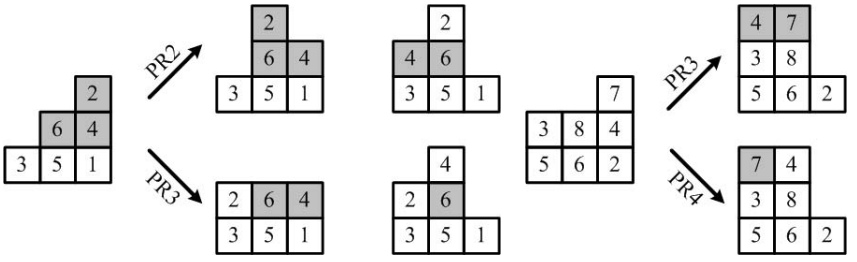
PR3:

a) If there are one or more stacks with $smallest(s)$ greater than the retrieval order of target container $c_{\bar{s},\bar{t}}$, select the stack with lowest $smallest(s)$;
b) Otherwise, select the stack with highest $smallest(s)$.

If there are multiple stacks with $smallest(s)$ greater than the retrieval order of container c, any such stack is an equally good choice for c. Therefore, we prefer the stack with the lowest $smallest(s)$ to maximize the number of containers that could satisfy this criterion in the future. Figure 5(a) illustrates the motivation for case a) for PR3 when compared to PR2, where containers that require at least one additional relocation are shaded.

PR4: Similar to PR3, except in case b) we perform an additional check: if the container to be relocated is not the 2nd smallest in the source stack, and the stack with highest $smallest(s)$ has only one available slot, then pick the stack with the 2nd highest $smallest(s)$ instead.

PR4 is motivated by the example illustrated by Figure 5(b). No matter which stack is selected as the destination for container 7, it must be relocated again. If we follow case b) of PR3 and choose stack 2 for container 7, then container 4 has to be relocated to stack 1, and at least one more relocation is needed for container 4. However, if we follow PR3 and pick stack 1 (the stack with 2nd highest $smallest(s)$) for container 7, then container 4 can be relocated to stack 2 and no future relocation is needed.



(a) PR2 versus PR3 (b) PR3 versus PR4

Fig. 5. Comparison of Probe Heuristics

6 Experiments and Analysis

Each test case is composed of three parameters N, S and T. A total of 125 sets of instances are randomly generated, with $S \in [6, 10]$, $T \in [3, 7]$ and $N \in [(S-1) \times T, S \times T - 1]$; 100 instances were generated for each combination of S, T and N. Note that a layout has solutions if and only if any container with retrieval order i has at most $i - 1 + e$ containers over it, where $e = S \times T - N$ is number of empty slots. This check only needs to be performed for containers with retrieval order $i = 1, 2, ..., T - e$; infeasible layouts are discarded and regenerated.

All experiments were conducted on a DELL personal desktop computer with a Pentium 4 CPU clocked at 3.0GHz, loaded with 1.0GB RAM and running the Windows XP operating system. All algorithms were implemented in C++ and compiled using Visual Studio C++ 2008 Team Edition (the default Release configuration for the Win32 platform was used).

The number of relocations estimated by lower bound LB2 is 4.26% larger than that of LB1 on average. The number of relocations estimated by lower bound LB3 is 10-20% larger than that of LB1. In general, the gap in performance between LB3 and LB1 increases as the size of the instances increase.

When the probe heuristics PR1, PR2, PR3 and PR4 are applied to the initial layouts of each test case, the average number of relocations (over all 12,500 cases) are 44.07, 37.87, 33.36, and 33.07 respectively. Since none of the heuristics strictly dominates all others over all test cases, we created a composite heuristic PR+ that simply takes the best solution found by PR1, PR2, PR3 and PR4.

Our best approach for solving the restricted variant using IDA* is as follows. We first construct an initial solution using heuristic PR+ (i.e., the best solution found by PR1, PR2, PR3 and PR4). During the search, LB3 is used as the lower bound to prune nodes, and heuristic PR4 is used to probe the children of frontier nodes. We refer to the resulting algorithm as IDA_PR+_LB3_PR4. We impose a strict time limit of 1 second of CPU time for each test instance; once the time limit is reached, the best solution found so far is reported.

To accurately gauge the performance of the algorithm, we first try to solve as many instances optimally as possible disregarding the time limit. We used a high performance computing server to run IDA_PR+_LB3_PR4 over a long period of

Table 1. Collated Results of IDA_PR+_LB3_PR4 for the Restricted Variant

Instance		Best Known		IDA_PR+_LB3_PR4				
T	Count	LB	Reloc	Reloc	LB Gap	Best Gap	time(s)	opt %
3	1500	13592	13592	13592	0.00%	0.00%	0	100.00%
4	2000	33095	33095	33095	0.00%	0.00%	0.12	100.00%
5	2500	63506	63507	63511	0.01%	0.01%	11.105	99.80%
6	3000	105549	105669	105918	0.35%	0.24%	117.729	92.63%
7	3500	160329	162660	164482	2.59%	1.12%	468.413	61.29%
Total	12500	376071	378523	380598	1.20%	0.55%	597.367	87.35%

time. For each instance we allow IDA_PR+_LB3_PR4 to explore as many as 2^{30} nodes.

Table 1 gives the collated results for the 12,500 instances. The results for all instances with same number of tiers are grouped together and the total number of relocations for the group is reported. *Count* shows the total number of instances in the group. The next two columns *LB* and *Reloc* gives the best known lower bounds and solutions, totaled over all instances in the group; the disparities between these two values arise when the algorithm is unable to find an optimal solution despite searching 2^{30} nodes.

The remaining five columns provide the results of IDA_PR+_LB3_PR4 under the 1 second time limit on a Pentium 4 desktop computer. *Reloc* gives the total number of relocations of the best solutions found by the algorithm for all instances in the group. This is followed by the percentage difference between this value and the best known lower bound (*LB Gap*), and between this value and the best known solutions (*Best Gap*), respectively. *time(s)* gives the total time in seconds elapsed before IDA_PR+_LB3_PR4 produced a solution, summed over all instances in the group; this value is somewhat inaccurate as we used the standard C library clock() function for this purpose, which has a precision of 1 millisecond, and many instances can be solved in less than 1ms. Finally, the last column *opt %* gives the percentage of instances in the group that was solved optimally, where the number of relocations of the solution matches the best known lower bound.

The data shows that the majority of instances with $T \leq 6$ can be solved optimally within 1 second. For large instances where $T = 7$, more than half the instances are solved optimally, and the average gap between the solution found and the best known lower bound is about 2.59%; this equates to an average difference from optimal of fewer than 2 relocations. We can conclude that our IDA_PR+_LB3_PR4 can solve instances of up to 7 tiers and 10 stacks to near optimal within 1 second for the restricted variant.

Our algorithm is a significant improvement over Kim and Hong [4], whose branch and bound approach is only able to solve instances of up to 5 tiers by 5 stacks within 4 minutes, and the solution found by the proposed heuristic is about 10% off the optimal solution.

Although Aydin [6] used a branch and bound algorithm similar to Kim and Hong [4] and is able to solve about 91% of instances of up to 7 tiers by 7 stacks, the test case generation strategies for that research have load factors ranging from 55% - 75%. Hence, the largest instance consists of only $7 \times 7 \times 0.75 = 37$ containers. In contrast, the test instances in our work are nearly fully loaded and consists of up to 69 containers. Since every subtree of the search tree with a root node containing m containers is itself a search tree, our algorithm actually solves several instances of the problem with $m = 37$ containers over the course of searching for solutions for larger instances.

7 Conclusions

In this paper, we examined the use of IDA* algorithms on the container relocation problem. We introduced two new lower bound measures for branch and bound pruning; in particular, the LB3 lower bound measure has proven to be much more effective than the LB1 measure that has been suggested in existing literature. We also made use of probe heuristics for promising partial solutions in order to narrow the search window; the heuristic PR3 has been found to be especially useful for this purpose. The resultant IDA_PR+_LB3_PR4 algorithm is able to solve instances of up to 7 tiers and 7 stacks to within 1 to 2 relocations of optimal for the restricted variant, which largely covers the set of practical cases. Hence, the restricted variant of the container relocation problem can be considered well solved for instances of practical size.

References

1. Ashar, A.: On selectivity and accessibility. Cargo Systems, 44–45 (June 1991)
2. Kim, K.H.: Evaluation of the number of rehandles in container yards. Comput. Ind. Eng. 32(4), 701–711 (1997)
3. Kim, K.H., Park, Y.M., Ryu, K.R.: Deriving decision rules to locate export containers in container yards. European Journal of Operational Research 124(1), 89–101 (2000)
4. Kim, K.H., Hong, G.P.: A heuristic rule for relocating blocks. Comput. Oper. Res. 33(4), 940–954 (2006)
5. Watanabi, I.: Selection process. Cargo Systems, 35–36 (March 1991)
6. Aydin, C.: Improved rehandling strategies for container retrieval process. Master's thesis, Graduate School of Engineering and Natural Sciences, Sabanci University (August 2006)
7. de Castillo, B., Daganzo, C.F.: Handling strategies for import containers at marine terminals. Transportation Research Part B: Methodological 27(2), 151–166 (1993)
8. Avriel, M., Penn, M., Shpirer, N., Witteboon, S.: Stowage planning for container ships to reduce the number of shifts. Annals of Operations Research 76(0), 55–71 (1998)