VARIABLE NEIGHBORHOOD SEARCH FOR THE BLOCK RELOCATION PROBLEM

Shunji Tanaka

Institute for Liberal Arts and Sciences, Kyoto University

tanaka@kuee.kyoto-u.ac.jp

ABSTRACT
In container yards, containers are stacked in several tiers due to space limitation. In order to retrieve a container from a container yard, it is necessary to relocate containers stacked on it. The aim of the block relocation problem, which is also known as the container relocation problem, is to minimize the number of relocations required for retrieving containers according to a specified order. This study will propose a variable neighborhood search algorithm for the problem and its effectiveness will be examined by numerical experiments.

Keywords: container terminal, block relocation problem, variable neighborhood search

1. INTRODUCTION

Container terminals in ports play an important role as a temporary storage for container transshipment between maritime and land transports. Containers in a container terminal are stacked in container yards to reduce space requirements. The containers compose multiple bays in a container yard, and each bay consists of several stacks. Containers are retrieved by a gantry crane that travels between bays and within a bay (Fig. 1). Since only containers on the top of stacks are accessible from the crane, those above the target container should be relocated to other stacks before it is retrieved. This relocation should complete within a bay because the crane travel from one bay to another is time-consuming compared to that within a bay. The purpose of the block relocation problem, which is also known as the container relocation problem, is to minimize the number of relocations necessary to retrieve all containers in a bay according to a specified retrieval order.

Formally, the problem considered in this study is described as follows.

Suppose a container bay composed of S stacks whose maximum height (the maximum number of tiers) is restricted to T. Blocks (containers) are stored in tiers and the number of blocks in stack i is given by $N_i (\le T)$. The total number of blocks is denoted by $N = \sum_{i=1}^{S} N_i$. The block in the j th tier of stack i from the bottom is referred to as block (i, j). Each block (i, j) is given a distinct integer priority P_{ij}

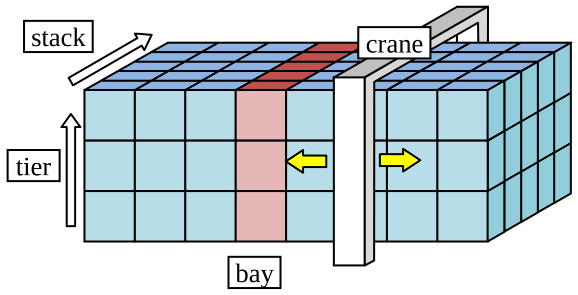

Figure 1: Containers in Container Yard

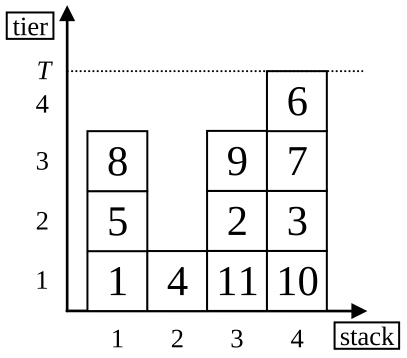

Figure 2: An Example of Block Layout ($S = 4, T = 4, N_1 = N_3 = 3, N_2 = 1, N_4 = 4$)

$(1 \le P_{ij} \le N)$ where a smaller value means a higher priority (an earlier retrieval order). Figure 2 illustrates an example of a block layout with $S = 4$ and $T = 4$. The number in each block denotes the priority P_{ij}.

The following two operations are available for retrieving all the blocks from the bay according to their priorities:

1. Relocation
   A block on the top of a stack is moved to the top of another stack whose height is less than T.
2. Retrieval
   A block with the highest priority is retrieved from the bay if it is on the top of a stack.

The objective of the block relocation problem is to find an optimal sequence of these two operations that minimizes the number of operations required. In practice, the objective function of the block relocation problem is the number of relocation operations because the number of retrieval operations is identical to the number of blocks and hence is constant.

There are two types of problem settings for the block relocation problem. In the restricted problem, the relocatable blocks are restricted to the one on the top of the stack that includes the block retrieved next, i.e. the block with the highest priority. In the layout of Fig. 2, block (1, 3) is the unique relocatable block because block (1, 1) has the highest priority and should be retrieved next. On the other hand, such a restriction is not imposed on relocatable blocks in the unrestricted problem. Hence, blocks (1, 3), (2, 1), (3, 3), and (4, 4) are all relocatable in this problem. This study will consider the restricted problem. In the following, the highest priority of the blocks in stack i, namely, $Q_i = \min_{1 \le j \le N_i} P_{ij}$ is referred to as the priority of stack i.

There have been several studies on heuristic algorithms for the block relocation problem (eg. Caserta and Voβ 2009; Caserta and Voβ 2011; Forster and Bortfeldt 2012; Petering and Hussein 2013; Jin et al. 2013). However, to the best of the author's knowledge, algorithms based on local search have not been studied extensively so far. The purpose of this study is to construct a local search algorithm and to examine its effectiveness by numerical experiments.

2. PROPOSED ALGORITHM

The proposed algorithm is a variant of the variable neighborhood search (Mladenović and Hansen 1997; Hansen et al. 2010). The constructive greedy heuristics by Caserta et al., 2011 is utilized for both computing the initial solution and representing solutions.

2.1. Constructive Heuristics

Suppose that the relocatable block in the current block layout is block (i, j) $(i = \arg\min_k Q_k)$ .This heuristics relocates block (i, j) to stack k^* determined as follows:

1. If there exists a stack whose height is less than T and whose priority is lower (larger) than P_{ij}, $k^* = \arg\min_{N_k < T, P_{ij} < Q_k} Q_k$.
2. Otherwise, $k^* = \arg\max_{N_k < T} Q_k$.

In the block layout in Fig. 2, the relocatable block (1, 3) is relocated to stack 2 according to the first condition.

2.2. Solution Representation

In the proposed algorithm, a solution is represented by destination stacks of relocatable blocks, which form a $(N-1)\times(T-1)$ matrix $X = (x_{ij})$. Each element x_{ij} is associated with a relocatable block as follows. Suppose that all the blocks with priorities higher than i have already been retrieved and that the block with priority i is block (k,l) $(P_{kl} = i)$. Since this block is to be retrieved next, the blocks above it should be relocated to other stacks, whose destination stacks are determined by x_{ij} . More specifically, the destination stack of the j th block $(k, l + j)$ is determined by x_{ij} .

Instead of encoding the destination stack directly, the proposed method encodes into x_{ij} the relative position of the destination stack in order to take advantage of the constructive heuristics. More specifically, the candidate destination stacks (the stacks other than stack k whose heights are less than T) are sorted in the increasing order of their priorities, and the positions of the destination stacks are denoted by their differences from that of the stack determined by the constructive heuristics. In the block layout of Fig. 2, for example, the block to be retrieved next is block (1, 1) and the relocatable block is (1, 3), which is associated with x_{12} . The candidates for its destination stack are stacks 2 and 3 whose priorities are 4 and 2, respectively. Therefore, 3, 2 is the sorted sequence of stacks. Since the destination stack determined by the constructive heuristics is stack 2, x_{12} should be either -1 or 0, which means that the destination stack is stack 3 or stack 2, respectively. Obviously, a zero solution matrix in this encoding always represents the solution by the constructive heuristics. Therefore, it will be easy to search around the heuristic solution intensively under this encoding method.

2.3. Variable Neighborhood Search

Based on the representation of a solution explained in the preceding subsection, a variable neighborhood search algorithm is applied. The initial solution is computed by applying a local search from the solution obtained by the constructive heuristics. The neighborhood of the incumbent solution in the local search is defined by those generated by changing the value of an element of the solution matrix. The incumbent solution is updated by the first improvement rule: it is updated immediately when a better solution is found in the neighborhood.

Next, the incumbent solution is perturbed randomly. The following three types of perturbations are employed in turns:

1. Insertion/Deletion
   An element is inserted into or deleted from a row of the solution matrix with an equal

probability of 0.5. In the case of insertion, an element with zero is inserted.

2. Modification
   The value of an element x_{ij} is changed randomly.
3. Interchange
   Values of two elements x_{ij_1} and x_{ij_2} in the same row are interchanged.

One of these perturbations is applied to randomly chosen rows and/or positions. Then, the local search is started again from this new incumbent solution. After these are repeated for some number of iterations, the incumbent solution is replaced by the current best solution, and then it is perturbed so that the local search can be started again from it.

It is possible that no destination stack exists for some value of x_{ij} . In this case, the feasibility of a solution is ensured by modulo operation.

The pseudocode of the proposed algorithm is summarized in Fig. 3. InsertionDeletion(X , p) in line 8 perturbs the solution matrix X by the insertion and deletion operations, and p specifies the total number of insertion and deletion operations applied, which is given by $p\lceil N/8\rceil$. Similarly, Modification(X , p) and Interchange(X , p) apply the modification and interchange operations, respectively, and the total numbers of operations applied are $p\lceil N/16\rceil$ and $p\lceil N/4\rceil$, respectively. After these perturbations are applied $9\,\mathrm{nrepeat}$ times, the incumbent solution is replaced by the best solution in line 23. If the best solution is not updated successively for 8 times, $\mathrm{nrepeat}$ is increased by 1, whose maximum value is restricted to 6.

All the parameters in the algorithm were determined by preliminary experiments.

3. NUMERICAL EXPERIMENTS

The proposed algorithm was applied to the 12500 benchmark instances by Zhu et al. (2012). The algorithm was coded in C and the experiments were conducted on a desktop computer with an Intel Core i7-2700K CPU (3.5GHz) by changing the maximum number of iterations ($\mathrm{maxiter}$) from 100 to 5000. To examine the effectiveness of the algorithm, the results are compared with optimal or best solutions obtained by the exact algorithms (Zhu et al. 2012; Tanaka and Takii 2014).

Table 1 summarizes the results. T and S denote the maximum height of stacks and the number of stacks, respectively, and n denotes the number of instances. “obj” and “time” are the average objective value and the average CPU time in seconds, respectively. In addition, “best” denotes the average objective value of optimal or best solutions found by the exact algorithms (Zhu et al. 2012; Tanaka and Takii 2014) as well as the proposed

```
1: Obtain an initial solution matrix $X^{\mathrm{incumbent}}$.
2: $X^{\mathrm{best}} \leftarrow X^{\mathrm{incumbent}}$, $i \leftarrow 1$.
3: $\mathrm{nrepeat} \leftarrow 1$, $\mathrm{notupdated} \leftarrow 0$.
4: while True do
5:   for $j = 1$ to $\mathrm{nrepeat}$ do
6:     for $k = 1$ to 9 do
7:       case $\lceil k/3 \rceil$ of
8:         1: InsertionDeletion( $X^{\mathrm{incumbent}}$, $k$ )
9:         2: Modification( $X^{\mathrm{incumbent}}$, $k-3$ )
10:        3: Interchange( $X^{\mathrm{incumbent}}$, $k-6$ )
11:      endcase
12:      Apply the local search from $X^{\mathrm{incumbent}}$.
13:      if $X^{\mathrm{incumbent}}$ is better than $X^{\mathrm{best}}$ then
14:        $X^{\mathrm{best}} \leftarrow X^{\mathrm{incumbent}}$
15:        $\mathrm{notupdated} \leftarrow -1$
16:      endif
17:      if $i = \mathrm{maxiter}$ then
18:        Output $X^{\mathrm{best}}$ and terminate.
19:      endif
20:      $i \leftarrow i + 1$
21:    endfor
22:  endfor
23:  $X^{\mathrm{incumbent}} \leftarrow X^{\mathrm{best}}$
24:  $\mathrm{notupdated} \leftarrow \mathrm{notupdated} + 1$
25:  if $\mathrm{notupdated} = 8$ then
26:    $\mathrm{notupdated} \leftarrow 0$
27     $\mathrm{nrepeat} \leftarrow \max(\mathrm{nrepeat} + 1, 6)$
28:  endif
29: endwhile
```

Figure 3: Pseudocode of the Algorithm

algorithm, and “heur” the average objective value of the constructive heuristics in 2.1, “ini” the average objective value of the initial solution (the solution obtained by applying the local search from the solution shown in “heur”. Boldface in the “obj” columns means that all the solutions yield the best objective values. The average CPU time for the initial solution is omitted because it was less than 0.01s. From this table, we can observe that the proposed algorithm is able to find good solutions quickly. Indeed, the best solutions for 85 instances among 180 unsolved instances by the exact algorithms were updated by the proposed algorithm.

4. CONCLUSION

This study proposed a variable neighborhood search algorithm for the block relocation problem. Numerical experiments showed that the algorithm is able to find good solution in a short time. We will be able to improve the performance of the local search in more sophisticated frameworks such as the tabu search algorithm, but it is left for future research. Extending the algorithm for the unrestricted problem is also left for future research.

Table 1: Computational Results for the Instances by Zhu et al. (2012)

T	S	n	best	heur	ini	100		500		1000		5000	
						obj	time	obj	time	obj	time	obj	time
3	6	300	**6.64**	6.71	**6.64**	**6.64**	0.00	**6.64**	0.01	**6.64**	0.02	**6.64**	0.06
	7	300	**7.77**	7.85	7.78	**7.77**	0.00	**7.77**	0.02	**7.77**	0.02	**7.77**	0.09
	8	300	**8.93**	9.01	8.93	**8.93**	0.01	**8.93**	0.02	**8.93**	0.03	**8.93**	0.14
	9	300	**10.37**	10.46	**10.37**	**10.37**	0.01	**10.37**	0.03	**10.37**	0.04	**10.37**	0.20
	10	300	**11.59**	11.73	11.60	**11.59**	0.01	**11.59**	0.03	**11.59**	0.06	**11.59**	0.27
4	6	400	**12.51**	12.91	12.57	**12.51**	0.01	**12.51**	0.02	**12.51**	0.03	**12.51**	0.13
	7	400	**14.50**	14.96	14.57	14.51	0.01	**14.50**	0.03	**14.50**	0.05	**14.50**	0.21
	8	400	**16.72**	17.35	16.83	16.73	0.01	**16.72**	0.04	**16.72**	0.07	**16.72**	0.32
	9	400	**18.46**	18.98	18.53	**18.46**	0.02	**18.46**	0.05	**18.46**	0.10	**18.46**	0.49
	10	400	**20.54**	21.21	20.69	20.55	0.02	**20.54**	0.07	**20.54**	0.14	**20.54**	0.71
5	6	500	**19.01**	20.24	19.26	19.04	0.01	19.01	0.03	19.01	0.05	**19.01**	0.25
	7	500	**22.52**	24.01	22.86	22.56	0.01	22.53	0.05	22.52	0.09	**22.52**	0.41
	8	500	**25.73**	27.45	26.10	25.78	0.02	25.73	0.07	25.73	0.14	**25.73**	0.68
	9	500	**28.31**	30.16	28.81	28.38	0.02	28.33	0.10	28.31	0.20	**28.31**	0.99
	10	500	**31.45**	33.44	31.98	31.53	0.03	31.47	0.14	31.45	0.28	**31.45**	1.43
6	6	600	**26.96**	29.68	27.67	27.09	0.01	26.98	0.05	26.98	0.09	**26.96**	0.45
	7	600	**31.00**	34.27	31.91	31.16	0.02	31.04	0.08	31.02	0.15	31.01	0.76
	8	600	**35.31**	38.89	36.30	35.54	0.03	35.38	0.12	35.34	0.24	35.32	1.20
	9	600	**39.52**	43.49	40.70	39.82	0.04	39.62	0.18	39.58	0.36	39.54	1.85
	10	600	**43.31**	47.80	44.66	43.68	0.05	43.46	0.25	43.40	0.51	43.34	2.62
7	6	700	**35.45**	40.29	36.94	35.83	0.02	35.56	0.08	35.51	0.15	35.46	0.74
	7	700	**41.10**	46.94	42.92	41.58	0.03	41.30	0.12	41.24	0.24	41.17	1.23
	8	700	**46.25**	52.80	48.33	46.87	0.04	46.51	0.20	46.43	0.40	46.33	2.06
	9	700	**51.93**	59.16	54.33	52.64	0.06	52.26	0.29	52.17	0.59	52.05	3.04
	10	700	**57.04**	65.14	59.72	57.91	0.08	57.49	0.43	57.37	0.86	57.19	4.49

REFERENCES

Caserta, M., Voβ, S., 2009. Corridor selection and fine tuning for the corridor method, *Lecture Notes in Computer Science*, 5851: 163–175.

Caserta, M., Voβ, S., Sniedovich, M., 2011. Applying the corridor method to a blocks relocation problem, *OR Spectrum*, 33: 915–929.

Forster, F., Bortfeldt, A., 2012. A tree search procedure for the container relocation problem, *Computers & Operations Research*, 39: 299–309.

Petering, M.E.H., Hussein, M.I., 2013. A new mixed integer problem and extended look-ahead heuristic algorithm for the block relocation problem, *European Journal of Operational Research*, 231: 120–130.

Jin, B., Lim, A., Zhu, W., 2013. A greedy look-ahead heuristic for the container relocation problem, *Lecture Notes in Computer Science*, 7906: 181–190.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search, *Computers & Operations Research*, 24: 1097–1100.

Hansen, P., Mladenović, N., Pérez, J.A.M., 2010. Variable neighborhood search: methods and applications, *Annals of Operations Research*, 175: 367–407.

Caserta, M., Schwarze, S., Voβ, S., 2012. A mathematical formulation and complexity considerations for the blocks relocation problem, *European Journal of Operational Research*, 219: 96–104.

Zhu, W., Qin, H., Lim, A., Zhang, H., 2012. Iterative deepening A^* algorithms for the container relocation problem, *IEEE Transactions on Automation Science and Engineering*, 9: 710–722.

Tanaka, S., Takii, K., 2014. A faster branch-and-bound algorithm for the block relocation problem, *to be presented at 2014 IEEE International Conference on Automation Science and Engineering*, August 18-22, Taipei (Taiwan).