Planning for Remarshaling in an Automated Container Terminal using Cooperative Coevolutionary Algorithms

Kiyeok Park
nerissa79@pusan.ac.kr

Taejin Park
parktj@pusan.ac.kr

Kwang Ryel Ryu
krryu@pusan.ac.kr

Department of Computer Engineering
Pusan National University
San 30, Jangjeon-dong, Geumjeong-gu, Pusan, 609-735, Korea

ABSTRACT

The productivity of a container terminal is highly dependent on the efficiency of loading the containers onto the vessels. The efficiency of container loading depends on how the containers are stacked in the storage yard. Remarshaling refers to the preparatory task of rearranging the containers to maximize the efficiency of loading. In this paper, we propose cooperative coevolutionary algorithms (CCEAs) to derive a plan for remarshaling in an automated container terminal. CCEAs efficiently search for a solution in a reduced search space by decomposing a problem into subproblems. Our CCEA decomposes the problem into two subproblems: one for determining where to move the containers and the other for determining the movement priority. Simulation experiments show that our CCEA can derive a better plan in terms of the efficiency of both loading and remarshaling than other methods which are not based on the notion of problem decomposition.

Categories and Subject Descriptors

I.2.8 [Problem Solving, Control Methods, and Search]: Scheduling

General Terms

Algorithms

Keywords

Container Terminal; Remarshaling; Cooperative Coevolutionary Algorithms

1. INTRODUCTION

The productivity of a container terminal is mainly dependent on the efficiency of loading the containers onto the vessels at the quay. The efficiency of container loading depends on how the containers are stacked in the storage yard, where containers are temporarily stored. Containers are loaded onto the vessels according to a predetermined plan, considering the weight balance of the vessel and the convenience of operations at the intermediate and the final destination ports. If a container picked for loading is stored under some other containers, additional operations are required for the yard crane to relocate the containers above it. This rehandling is the major source of inefficiency of loading, causing delays at the quay. Loading operation is also delayed if a yard crane needs to travel a long distance to fetch a container for loading.

Export containers delivered to the terminal are temporarily stored in the storage yard for one or two weeks prior to shipping. During this period, it is difficult to stack the containers in an ideal configuration enabling efficient loading. The reason is that various types of containers to be shipped to different vessels must be stored together in the yard with limited capacity. Moreover, until a few hours before the loading starts, the precise container weights are not yet informed and the loading plan is not yet determined, either [5]. Remarshaling is the preparatory task of rearranging the containers so that the yard crane does not need to travel a long distance and does not need to do rehandling work at the time of loading.

This paper proposes a method that plans for remarshaling in an automated container terminal using cooperative coevolutionary algorithms (CCEAs) [3]. CCEAs efficiently search for a solution in a reduced search space by decomposing a given problem into subproblems [10]. In CCEAs, there is a population of candidate solutions for each subproblem, and these populations cooperatively evolve via mutual information exchanges. In our CCEAs, the problem of remarshaling is decomposed into two subproblems: one for determining the target slots to which the containers are moved and the other for determining the priority of moving those containers. Simulation experiments show that our CCEA constructs a better plan than other methods not based on the concept of problem decomposition, in terms of the efficiency of both loading and remarshaling. There are previous studies that solve the problem of remarshaling from various perspectives by using various methods [2,4,6,8,12]. This work, however, gives solutions for conventional container terminals where the layout of storage yard is different from that of the automated container terminal that we are targeting. Since different yard layout entails different objectives of remarshaling, those solutions are not applicable to our target terminal. More importantly, none of them adopts the notion of problem decomposition and coevolution.

In the next section, we explain our target problem in more detail. In Section 3, we describe the method of planning for remarshaling using CCEAs. In Section 4, we analyze and discuss results of simulation experiments. Finally in Section 5, we give some concluding remarks.

2. Remarshaling in an Automated Container Terminal

Figure 1 illustrates our target automated container terminal that can be largely divided into three areas: the quayside where vessel operations are performed by the quay cranes (QCs), the storage yard where containers are temporarily stored, and the hinterland where external trucks are driven. The delivery of containers between the quayside and the storage yard is done by the automated guided vehicles (AGVs). In the storage yard, there are typically tens of blocks laid out in perpendicular to the quayside. As shown in more detail in Figure 2, each block consists of tens of bays and each bay consists of several stacks of containers piled into a number of tiers. A block in our target terminal has 41 bays, and each bay has 10 stacks with a maximum of 5 tiers. The container operations at a block are done by two automated transfer cranes (ATCs). One of them performs seaside operations and the other performs hinterland operations.

There are four major operations in a container terminal: loading, discharging, carry-in, and carry-out. Loading operations involve shipping export containers that have been stored in the yard. The discharging operations involve unloading import containers from a vessel. The carry-in operations involve storing the export containers delivered from outside into the storage yard. The carry-out operations involve retrieving the import containers that have been stored in the storage yard to transport out of the terminal. Note that the loading and discharging operations need cooperation of QCs, ATCs, and AGVs, and the carry-in and carry-out operations require interfacing of ATCs with external trucks.

The productivity of a container terminal depends on the efficiency of loading operation executed by the QCs that are the most critical resources in the terminal. For a QC to load a container onto a vessel, an ATC must first fetch a container from a block and put it onto an AGV before the AGV delivers it to the QC. Accordingly, the operation of QC is delayed if any of the operations by ATC or AGV is delayed. Remarshaling is intended for elimination of the possibility of delays by ATC operations, thereby enhancing the efficiency of loading.

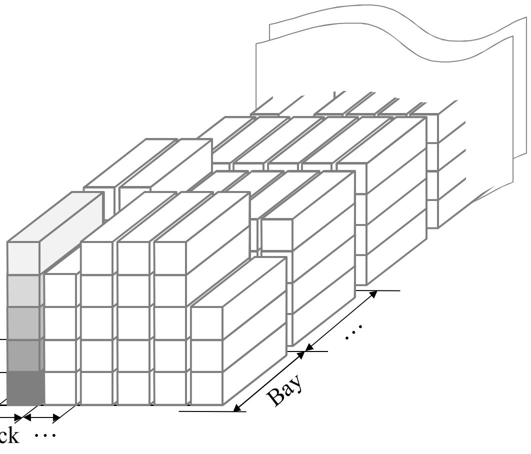


Figure 2. Detailed view of a block

In a block that is laid out in perpendicular to the quayside, as in an automated container terminal, ATC operations are delayed in two ways. The first is when an ATC has to travel a long distance to pick up a container for loading. Export containers delivered by external trucks are mostly piled near the hinterland side of a block. Storing an export container near the seaside of a block on delivery may result in interference between the landside and seaside ATCs, thus disrupting the seaside ATC while it processes vessel (loading or discharging) operations. Since the disruption of vessel operations directly hurts the productivity of the container terminal, it is safe for the landside ATC to store the carried-in containers near the hinterland side of a block at the time of delivery. This, however, causes the seaside ATC to travel a long distance at the time of loading the container. The second is when rehandling situation arises at the time of loading due to the inappropriately stacked containers mismatching the loading sequence. As mentioned previously, it is hard to store all the carried-in containers in ideal

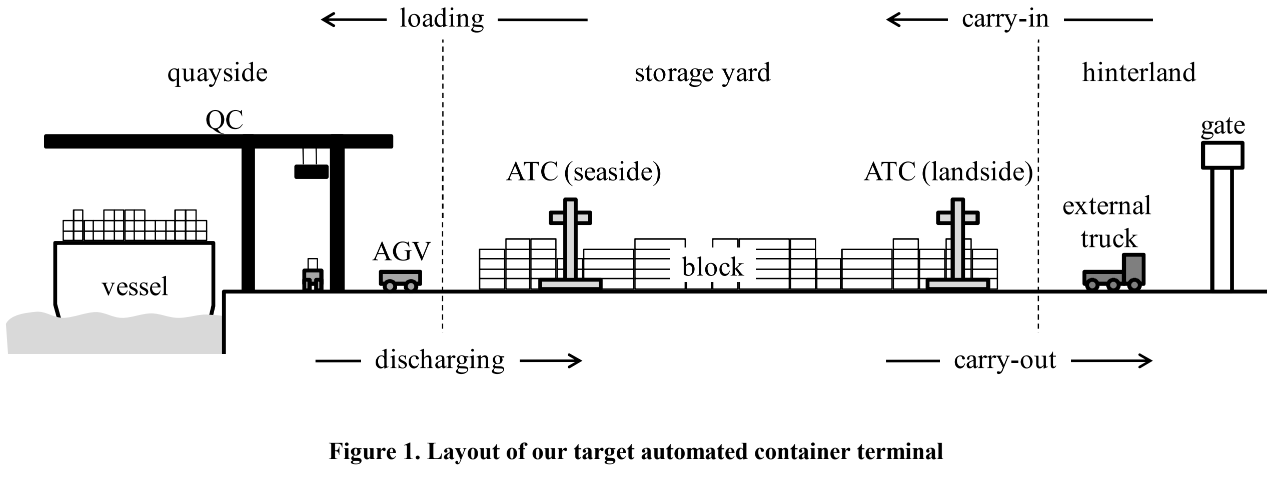


Figure 1. Layout of our target automated container terminal

slots due to the space limitation, imprecise weight information, and other restrictions. Remarshaling reduces the possibility of ATC delays as much as possible by reshuffling the containers to respect the loading sequence, by moving the export containers from the landside to the seaside, and also by moving some of the import containers from the seaside to the landside.

3. DERIVING A PLAN FOR REMARSHALING

We build a plan for remarshaling in two stages. In the first stage, we use heuristics to select candidate stacks to relocate the containers that will be remarshalled. The stacks near the seaside end of a block are obvious candidates because the travel distance of the seaside ATC will be short for loading. More conditions for candidate stacks are given below in section 3.1. Once the candidate stacks are selected, a specific target slot is assigned to each of the containers to be remarshalled. If the target slots are determined appropriately, the rehandlings are completely avoided during loading. Also, well-determined target slots minimize the rehandling during remarshaling. One last thing to be determined is the order of movement of the containers to be remarshalled. If the containers are moved in such an order that the rehandlings are also avoided during remarshaling, the time spent for remarshaling operation itself will be minimized. Therefore, in the second stage, we use a cooperative coevolutionary algorithm (CCEA) to select the target slots as well as the movement priorities for the containers to be remarshalled. Our CCEA evolves in parallel two different populations each for finding the target slots and determining the movement priorities. The cooperation through information exchange between the two populations takes place when the individuals of the populations are evaluated. An individual of one population is evaluated by being combined with a collaborator in the other population. How we apply a CCEA to our problem is described in detail in section 3.2.

3.1 Heuristic selection of candidate stacks for relocating the containers

We heuristically select the candidate stacks where the containers to be remarshalled are newly placed. While selecting the candidate stacks, we consider a few constraints to maximize the efficiency of loading.

First, the candidate stacks must be close to the seaside having at least one empty slot or removable container on top. The removable containers are those that have been discharged and are waiting to be carried out by external trucks. Removing these containers not only secures spaces for remarshaling but also brings about other positive side effects. When we remove them, we move them to locations toward the hinterland side of the block because the goal of removal is to leave empty spaces at the seaside. As they are moved more toward the hinterland side, the carry-out operations will be more expedited. Note that removable containers do not include the containers that will be loaded onto other future vessels. Removing such containers merely results in additional remarshaling works to move them back to the seaside for future loading.

Second, the export containers that will be loaded by different QCs must be piled in different stacks. Often, multiple QCs work in parallel to load a vessel and each QC follows a predetermined loading sequence of containers. There is no ordering constraint between the containers loaded by different QCs and we do not really know which of them will be loaded before the others. Therefore, rehandlings can arise if such containers are mixed together in the same stack. To prevent such rehandlings, our heuristic selects candidate stacks separately for each QC.

Third, the containers to be loaded by different QCs must be placed in the stacks at a similar average distance from the seaside ends of the blocks. For example, consider the case in which two QCs are loading containers stored in several blocks. If the containers of QC A are stored in the stacks at longer average distance from the seaside ends than the containers of QC B, the progress of QC A will be slower than that of QC B because the average travel distance of ATCs is longer for the containers of QC A. Since the overall completion time for loading is determined by the slowest QC, it is the best if we can have all the QCs finish at the same time.

Figure 3 shows an example of candidate stacks selected by our heuristic. Assuming that there are three QCs for loading, the figure shows three stacks A_1, B_1, and C_1 that are selected as the first stacks for the QCs A, B, and C, respectively. Stack A_1 has four empty slots with one leftover container at the bottom that is not removable because it will later be loaded onto a different vessel. Stack B_1 has all of its slots empty. Stack C_1 has three empty slots with two discharged containers at its bottom that will be removed during remarshaling. Recall that a stack is selected as a candidate if it has at least one empty slot.

Figure 4 shows the algorithm for selecting candidate stacks. Initially, for each QC q in Q that is a circular queue of QCs, we calculate the number n_q of containers that will be loaded by q. This number n_q is then immediately multiplied by $(1+\alpha)$ where $\alpha \geq 0$. The reason for this is to provide extra rooms for remarshaling so that more rehandlings can be avoided during the operation of remarshaling. The value for α was empirically set to 0.1. Then, for the first QC q in Q, we select a stack satisfying both the first and the second constraints mentioned above and add it to T_q, the set of candidate stacks for the QC q. If the number of usable slots in T_q is greater than or equal to n_q, then we stop selecting stacks for q and continue with the next QC. Subsequent to selecting a stack for

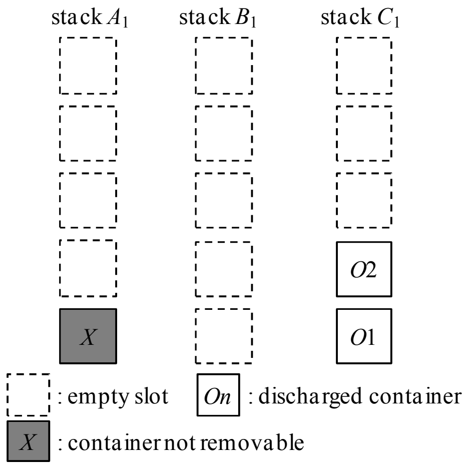


Figure 3. An example of candidate stacks

```
Q: circular queue of QCs
T_q: set of candidate stacks for QC q  (initially empty)

1. For each QC q in Q
     $n_q \leftarrow$ the number of containers to be loaded by q
     $n_q \leftarrow (1 + \alpha) \times n_q$
2. Repeat the following steps (2.1~2.4) until enough slots are
   selected for each QC:
     2.1 $q \leftarrow$ next one in Q
     2.2 Go back to step 2.1 if
             (the number of usable slots in $T_q$) $\geq n_q$
     2.3 Select an appropriate candidate stack t
     2.4 $T_q \leftarrow T_q \cup \{t\}$
```

Figure 4. Algorithm for selecting candidate stacks

the last QC in the circular queue Q, we return to the first QC in Q and start selecting another stack for it. This selection procedure is repeated until the number of usable slots in T_q is greater than or equal to n_q, for every q in Q. Notice that this round-robin selection guarantees the satisfaction of the third constraint mentioned above.

3.2 Determination of target slots and movement priority using CCEA

CCEAs decompose a given problem into multiple sub-problems to reduce the search space, and evolve a population of solutions for each of the subproblems [3]. In this paper, we decompose our problem into two sub-problems for the CCEAs to search efficiently: one is determining target slots for relocating containers and the other is determining movement priority for minimizing remarshaling time by avoiding rehandling during remarshaling.

The subproblem of determining target slots involves determining which container is placed in which slot of the candidate stacks selected by the heuristics of section 3.1. We generate a candidate solution for this subproblem in the following way. First, we separate the containers that will be loaded by different QCs. Then, for each QC, we randomly select a container of that QC and then randomly assign it to a slot of the candidate stacks of that QC. Since there are some extra stacks reserved than necessary as described in section 3.1, some of the slots remain unoccupied. Figure 5(a) illustrates a candidate solution. It looks as if it is obtained by laying down horizontally the empty slots of the stacks of Figure 3 and then randomly assigning the containers of the corresponding QCs. A rectangle with a bar ('—') indicates an unoccupied empty slot. For QC C, for example, four containers were assigned to stack C_1 presuming that the two containers $O1$ and $O2$ will be removed, leaving one slot unoccupied.

Figure 5(b) illustrates a candidate solution for the sub-problem of determining movement priority. This candidate solution is generated by arranging the containers in a random order. The containers in this solution include not only those that will be remarshalled but also the removable containers such as $O1$ and $O2$ of Figure 3. The destination slots for these removable containers are not specified in the plan of remarshaling but determined heuristically at run time when the plan is executed.

In order to evaluate a candidate solution of a sub-problem, we need to assess how much the information contained in that solution contributes to building a good remarshaling plan. If the subproblems resulting from problem decomposition are completely independent of each other, such an assessment may be made separately for each subproblem. If not, however, the evaluation of a candidate solution should be done by combining it with other candidate solutions of other subproblems. The candidate solutions collected for combination are called collaborators. In fact, the fitness value of a candidate solution of a subproblem depends on the collaborators it is combined with. However, trying all possible combinations to find the fittest collaborator demands high computational overhead. Therefore, some CCEAs maintain a separate set of promising collaborators, and only those are examined when finding the fittest collaborator [9,11]. Our CCEAs minimize the overhead by selecting, as a sole collaborator, the candidate solution of the highest fitness from the population of the other subproblem.

In our CCEAs, the evaluation of a candidate solution starts with the problem of determining target slots. From a candidate solution such as the one in Figure 5(a), we first construct a target configuration such as that of Figure 6(a) which guarantees no rehandling to occur during loading. This can be done by simply sorting the containers in each stack of Figure 5(a) separately in accordance with the loading sequence. After the sorting, we derive a partial order of containers as shown in Figure 6(b). This partial order is easily obtained by first tracking the order of removal of the containers in the stacks if any, and then following the bottom-to-top order of the containers in the stacks of the target configuration. In the example of Figure 6(b), we can see that the containers $O2$ and $O1$ must be removed from stack C_1 in that order, and then the containers $C8$, $C7$, $C5$, and $C1$ should be placed in the bottom-to-top order. If this partial order is followed during remarshaling, the

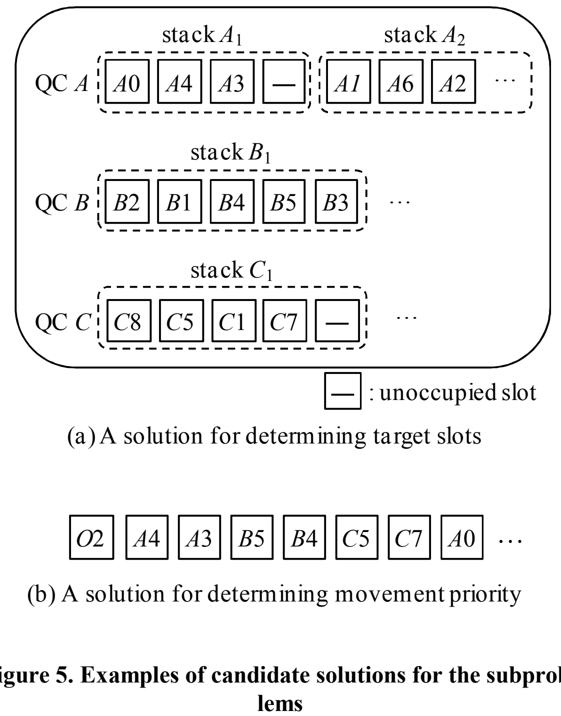
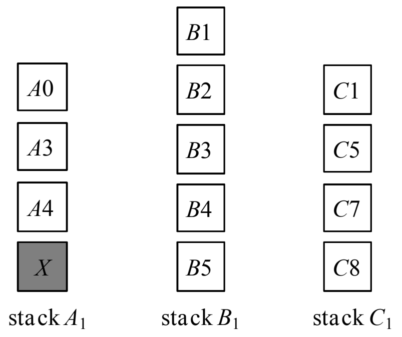
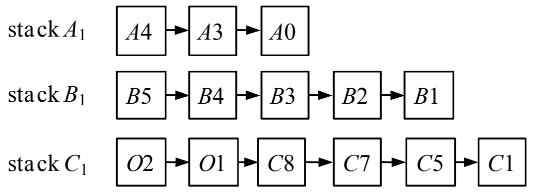


Figure 5. Examples of candidate solutions for the subproblems



(a) A target configuration



(b) The corresponding partial order

Figure 6. An example of a target configuration and the corresponding partial order

target configuration can be achieved in the most efficient way.

Given any target slot assignments, rehandling-free loading is ensured by sorting the containers in each stack as explained above. However, ill-determined target slots may cause rehandlings to occur during remarshaling. For example, the slot assignment of Figure 5(a) causes rehandling if the container *A*3 is stacked on top of *A*4 in their original locations before remarshaling. This rehandling is avoided if one of them is assigned to a target slot in a different stack, say A_2.

The partial order mentioned above only constrains the order of moving the containers belonging to the same target stack, but not those of different stacks. A total order of container movement can be derived by combining the partial order with a candidate solution of the subproblem of determining the movement priority. Suppose that the solution shown in Figure 5(b) is the collaborator of the partial order of Figure 6(b). Then, *O*2 becomes the first one to be moved among *A*4, *B*5, and *O*2 because it precedes the others in the movement priority. Subsequently, the tie among *A*4, *B*5, and *O*1 is broken and *A*4 is selected by referring to the movement priority again. Once a total order of container movement is determined in this manner, we can estimate via simulation the total time taken to finish remarshaling. The evaluation of a remarshaling plan is completed by adding the ATC loading delay to the remarshaling time just estimated. The ATC loading delay is predicted by calculating the total ATC travel time during loading. The loading delay is minimized if the containers are relocated to the slots as near the seaside end as possible.

The genetic operators used for evolving the population are order crossover [1] and swap mutation. For the sub-problem of determining movement priority, the order crossover is applied to two randomly selected candidate solutions in an ordinary way because each candidate solution is a simple ordered sequence. A solution is mutated by simply swapping the loci of two randomly selected containers. For the subproblem of determining target slots, the order crossover cannot be applied ordinarily because a candidate solution is a sequence of sub-solutions ($s_1, s_2, \ldots, s_k$) where s_i is again a sequence of target slots for the containers to be loaded by QC *i*. Two random solutions ($s_1, s_2, \ldots, s_k$) and ($t_1, t_2, \ldots, t_k$) being selected, the order crossover is applied to each pair of sub-solutions s_i and t_i. A solution is mutated by applying the swap mutation to each sub-solution, but under the condition that only the containers belonging to different stacks can be swapped.

4. Experiments

In this section, we first describe the algorithms of generational and steady-state CCEA models that we have experimented with. Then, we explain our experimental setting and report the results with some discussions.

4.1 Algorithms tested

CCEAs are classified into parallel and sequential types depending on how the populations are updated [3]. While sequential CCEAs evolve each population in turn in each generation, parallel CCEAs evolve all populations simultaneously in each generation. The generational and steady-state CCEAs that we have experimented with are both parallel CCEAs.

With a generational CCEA, the individuals of a population of the current generation are completely eliminated when the population evolves to the next generation. Therefore, the fittest individual of each population of the current generation is reserved separately to be used as a collaborator in the next generation. The algorithm of the generational CCEA used in our experiment is given in Figure 7. The algorithm starts by constructing initial populations P_s and P_p for the subproblems of determining target slots and movement priority, respectively. Subsequently, initial collaborators c_s and c_p are randomly generated for evaluation of the initial populations P_p and P_s, respectively. After both populations are evaluated using the respective collaborators, the fittest individual from each popu-

```
1. Generate initial populations $P_s$ and $P_p$ for the slot and prior-
   ity determination subproblems, respectively.
2. Generate initial collaborators $c_s$ and $c_p$ for evaluation of the
   initial populations $P_p$ and $P_s$, respectively.
3. Repeat the following steps until the termination condition
   is satisfied:
  3.1. For each individual $i$ of $P_s$
     3.1.1. construct a schedule $S$ using $i$ and $c_p$
     3.1.2. calculate the fitness of $i$ by evaluating $S$
  3.2. For each individual $i$ of $P_p$
     3.2.1. construct a schedule $S$ using $i$ and $c_s$
     3.1.2. calculate the fitness of $i$ by evaluating $S$
  3.3. $c_s$ ← the fittest individual of $P_s$
       $c_p$ ← the fittest individual of $P_p$
  3.4 Update $P_s$ and $P_p$ by applying genetic operators
4. Return the best-so-far schedule
```

Figure 7. Algorithm for generational CCEA

lation is saved to serve as a collaborator in the next generation. Then, each population is genetically updated, and the evaluation and update steps repeat until the termination condition is met.

In a steady-state CCEA, whenever a new offspring is born, it goes through a survival selection with the individual of the lowest fitness in the population. Unlike the generational model, the fittest individual does not suddenly disappear from the population, and thus it is used as a collaborator for evaluating new offspring. The algorithm of our steady-state CCEA is given in Figure 8. Initial populations are generated in the same way as the generational model. However, evaluation of each individual in the initial populations is more thoroughly done by trying all the individuals of the counterpart population as candidate collaborators and taking the best value. The reason for this is that the individuals of the initial populations in a steady-state model survive for a relatively long duration, influencing the direction of search. From then on, the fittest individual of a population serves as a collaborator whenever a newly born offspring of the other population is evaluated.

4.2 Experimental setting

We assumed that the container terminal used for experiments was featured as follows. One vessel has docked at the quay and the containers that will be loaded onto the vessel are stored in seven different blocks. A block has 41 bays and each bay consists of 10 five-tier stacks. Although each block is equipped with two ATCs, only the seaside ATC is used for remarshaling. We prepared scenarios of three different scales each with 500, 1000, and 1500 containers to be loaded. We experimented with 10 different data sets for each scenario and averaged the results. The data sets were generated based on statistics from a real-world terminal.

In the experiments, the two CCEAs described above were compared with standard genetic algorithms of both the generational and steady-state models, which do not adopt the concept of problem decomposition. For fair comparisons, the number of evaluations for all the algorithms was limited to 50,000, 100,000 and 150,000 for the scenarios of 500, 1000, and 1500 containers, respectively. Binary tournament selection was used for parent selection in all the algorithms. Truncation selection was used for survival selection in the steady-state model.

```
1. Generate initial populations $P_s$ and $P_p$ for the slot and priority determination subproblems, respectively.
2. Evaluate each individual of $P_s$ and $P_p$ by considering all possible collaborators and take the best value.
3. Repeat the following steps until the termination condition is met:
  3.1. $c_s \leftarrow$ the fittest individual of $P_s$
       $c_p \leftarrow$ the fittest individual of $P_p$
  3.2. Produce an offspring for each population.
  3.3. For the new offspring $o$ of $P_s$
    3.3.1. construct a schedule $S$ using $o$ and $c_p$
    3.3.2. calculate the fitness of $o$ by evaluating $S$
  3.4. For the new offspring $o$ of $P_p$
    3.4.1. construct a schedule $S$ using $o$ and $c_s$
    3.4.2. calculate the fitness of $o$ by evaluating $S$
  3.5. For each population, the new offspring goes through a survival selection
4. Return the best-so-far schedule
```

Figure 8. Algorithm for steady-state CCEA

After trying various population sizes and mutation rates, we found that the generational CCEA (gCCEA) performed the best when the population size was 10 and the mutation rate was $1/L$, where L is the length of a chromosome. The steady-state CCEA (sCCEA) performed well with a population size of 10 and a mutation rate of 0.1. For the generational GA (gGA) , a population size of 10 and a mutation rate of $1/L$ were the best. For the steady state GA (sGA), a population size of 50 and a mutation rate of 0.1 were the best.

4.3 Results and discussions

Table 1 shows the evaluation value of the best solution found by each algorithm for each scenario. Recall that the evaluation value is the summation of the ATC travel time for loading and the estimated remarshaling time. While sGA was the best for the scenario with 500 containers, sCCEA was the best for the other scenarios with larger number of containers. Overall, the steady-state models showed better performance than the generational models for the given data sets, and the difference became larger as the problem size got bigger. Comparing standard GAs with CCEAs, GA outperformed CCEA with the generational model, and CCEA outperformed GA with the steady-state model.

Table 2 confirms us that the loading operation becomes much more efficient if the containers are remarshalled by executing the plan derived by our algorithms. The data in the table was obtained

Table 1. Evaluation values of the best solutions found by different algorithms.

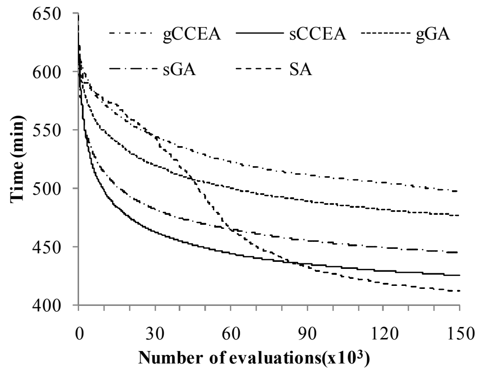
# of containers loaded	Algorithms	Best evaluation values (sec.)
500	gCCEA	14,692.5
	sCCEA	14,550.1
	gGA	14,693.9
	sGA	**14,492.0**
1000	gCCEA	30,702.2
	sCCEA	**28,936.8**
	gGA	30,504.3
	sGA	29,252.3
1500	gCCEA	50,275.3
	sCCEA	**44,884.9**
	gGA	48,542.3
	sGA	46,434.2

Table 2. Time taken for loading (hours)

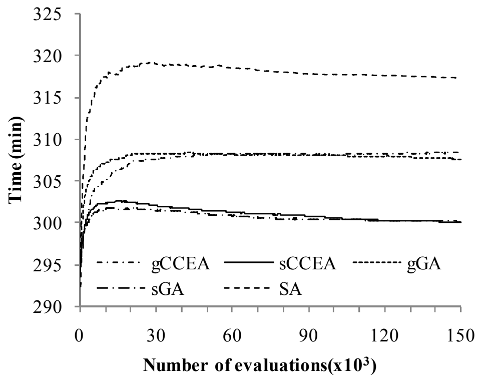
# of containers loaded	Without remarshaling	With remarshaling
500	5.90	5.09
1000	13.21	9.87
1500	20.85	14.86

via simulation based on a detailed modeling of stacking status of containers as well as crane movements. Depending on the algorithm used, the time taken for loading after remarshaling was a little different. The loading time in Table 2 is those obtained with the remarshaling plan derived by sCCEA. Recall that the candidate stacks for relocating the containers for remarshaling are selected by using the heuristics of section 3.1 before applying the search algorithms. As mentioned earlier in section 3.2, it is very easy to derive a remarshaling plan which guarantees rehandling-free loading. Therefore, the difference in loading efficiency comes only from the difference in the ATC travel distance. Recall again that our heuristics of section 3.1 reserve 10% extra space for the candidate stacks where the export containers are to be relocated. If the target slots are all concentrated towards the seaside end of the block leaving the extra space empty behind, maximum efficiency of loading can be achieved by minimizing the ATC travel distance. However, rehandlings may be harder to be avoided during remarshaling in this case. On the other hand, if the target slots are distributed evenly over the entire space reserved, it becomes easier to avoid rehandlings during remarshaling. However, the loading efficiency will be sacrificed.

Figure 9 shows the curves of the values of the two components of our evaluation function for the scenario with 1500 containers: the remarshaling time and the estimated ATC travel time for loading.



(a) Remarshaling time



(b) Estimated ATC travel time for loading

Figure 9. Curves of the values of the two components of the evaluation function for the scenario with 1,500 containers

The figure also includes curves obtained in additional experiments using a simulated annealing (SA) algorithm [7] without problem decomposition. All the curves are obtained by averaging the evaluation values of 10 different runs. We can see that the plans derived by SA tend to require less time for remarshaling but more time for loading. SA seems to sacrifice the loading time by selecting the target slots relatively uniformly over all the candidate stacks, thus expediting remarshaling by minimizing rehandlings. However, we observed that the best overall evaluation values of SA were worse than those of sCCEA in Table 1 for all the scenarios, although it might be still premature to conclude that SA should always perform worse.

5. Concluding Remarks

A contribution of this paper is the introduction of a method of planning for remarshaling in a perpendicularly laid-out storage block which is typical in automated container terminals. Since previous work on remarhsalling focuses on a horizontally laid-out block where the objectives sought for remarshaling are different, they are not easily adapted to a perpendicularly laid-out block. Another contribution of this paper is to confirm the effectiveness and efficiency of CCEAs in solving the problem of planning for remarshaling by adopting the notion of problem decomposition.

In fact, it is very natural to think of the remarshaling problem in terms of the two subproblems of determining target slots and movement sequence. Kang et al. [4] adopted this idea to solve the problem of remarshaling in a horizontally laid-out block, but took a hierarchical approach in which one subproblem is treated to be completely subordinate to the other. They thought the ultimate goal of the problem is to search for good target slots for the containers to be remarshalled. A candidate solution for determining target slots is evaluated by estimating the time taken to move the containers to their respective slots. For this, another search is conducted to find the best crane schedule to move the containers requiring the shortest time. The problem is that the overall search becomes infeasible if the search cost at the lower level is nontrivial. Therefore, they made compromise by approximating the lower level search for the crane scheduling.

In contrast, the CCEAs that we advocate in this paper decompose the remarshaling problem into two semi-independent subproblems at the same level. After finding a reasonably good solution for the subproblem of target slots, it is taken as a collaborator for finding a good solution for the subproblem of movement sequence. Subsequently, the good movement sequence just found becomes a collaborator for finding good target slots. This alternating search continues until a plan of satisfactory quality is found. As shown by the experimental results, the proposed method finds a better solution than other methods which do not adopt the notion of problem decomposition.

6. Acknowledgment

This research was supported by a grant from "Development of Intelligent Port Logistics System" Project funded by Ministry of Land, Transport and Maritime Affairs of Korean government.

7. References

[1] Davis, L., Applying adaptive algorithms to epistatic do-mains, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985, pp. 162-164.

[2] Hirashima, Y., Ishikawa, N., and Takeda, K., A new reinforcement learning for group-based marshaling plan considering desired layout of containers in port terminals, *Proc. IEEE Conf. Networking, Sensing and Control*, April 2006, pp. 670-675.

[3] Husbands, P., and Mill, F., Simulated coevolution as the mechanism for emergent planning and scheduling, *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 264-270.

[4] Kang, J., Oh, M. S., Ahn, E. Y., Ryu, K. R., and Kim, K. H., Planning for intra-block remarshalling in a container terminal, *IEA/AIE, LNAI 4031*, 2006, pp. 1211-1220.

[5] Kang, J., Ryu, K. R., and Kim, K. H., Determination of Storage Location for Incoming Containers of Uncertain Weight, *IEA/AIE, LNAI 4031*, 2006, pp. 1159-1168.

[6] Kim, K. H., and Bae, J. W., Re-marshaling export containers in port container terminals, *Computers and Industrial Engineering,. 35, 3-4*, 1998, pp. 655-658.

[7] Kirkpatric, S., Gelatt, C. D., and Vecchi, M. P., Optimization by simulated annealing*, Science, 220,* 1983, pp. 671-680.

[8] Lee, Y., and Hsu, N. Y., An optimization model for the container pre-marshalling problem, *Computer and Operations Research, vol. 34*, 2007, pp. 3295-3313.

[9] Panait, L., Luke, S., and Harrison, J. F., Archive-based Cooperative Coevolutionary Algorithms, *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006, pp. 345-352.

[10] Wiegand, R. P., *Analysis of Cooperative Coevolutionary Algorithms*, PhD thesis, Department of Computer Science, George Mason University, 2003.

[11] Wiegand, R. P., and Sarma, J., Spatial embedding and loss of gradient in cooperative coevolutionary algorithms, *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature*, 2004, pp. 912-922.

[12] Zhang, Y., Mi, W., Chang, D., and Yan, W., An Optimization Model for Intra-bay Relocation of Outbound Container on Container Yards, *International Conference on Automation and Logistics*, 2007, pp. 776-781.