J Intell Manuf (2011) 22:201–217
DOI 10.1007/s10845-009-0273-y

Generating a rehandling-free intra-block remarshaling plan for an automated container yard

Ri Choe · Taejin Park · Myung-Seob Oh · Jaeho Kang · Kwang Ryel Ryu

Received: 25 February 2008 / Accepted: 31 May 2009 / Published online: 13 June 2009

Abstract Intra-block remarshaling in a container terminal refers to the task of rearranging the export containers, which are usually scattered around within a block, into designated target bays within the same block. Since the containers must be loaded onto a ship following a predetermined order, the rearrangement should be performed in such a way that the containers to be loaded first are placed on top of those to be loaded later in order to avoid rehandling. To minimize the time required to complete a remarshaling task, rehandling should also be avoided during the remarshaling operations. Moreover, when multiple stacking cranes are used for the remarshaling, the interference between cranes should be minimized. This paper presents a method to efficiently search for an intra-block remarshaling plan which is free from rehandling during both the loading operation and remarshaling, and which minimizes the interference between the stacking cranes.

Keywords Multiple crane scheduling · Container terminal · Intra-block remarshaling · Simulated annealing

R. Choe · T. Park · M.-S. Oh · J. Kang · K. R. Ryu (✉)
Department of Computer Engineering, Pusan National University, Jangjeon-dong San 30, Geumjeong-gu, Busan 609-735, Korea
e-mail: krryu@pusan.ac.kr

R. Choe
e-mail: choilee@pusan.ac.kr

T. Park
e-mail: parktj@pusan.ac.kr

M.-S. Oh
e-mail: oms1226@pusan.ac.kr

J. Kang
e-mail: jhkang@pusan.ac.kr

Introduction

Background and motivation

The productivity of a container terminal primarily depends on the efficiency of the operation of loading the containers onto the vessels. When the containers are loaded they must follow some ordering constraints imposed by their size, port of destination, weight, and so on. The containers destined for more distant ports have to be loaded before those destined for closer ports because these should be stowed in the upper part of the vessel holder or deck. Also, heavier containers must be loaded before lighter ones because the heavier ones should be stowed in the lower part of the vessel holder to provide greater stability. All these constraints are taken into account when the loading sequence is determined prior to the loading operation. To obtain a good loading sequence, however, one must also carefully examine the storage locations of the individual containers in the yard. There are many blocks in the yard and each block typically accommodates around a thousand containers and is equipped with one or two stacking cranes (SCs). If the containers that are stacked far apart in a block are placed at successive positions in the loading sequence, the SC is forced to travel a long distance, thus spending an excessive amount of time. Also, if a container is stacked under some others, but is at an earlier position in the sequence than those above it, the SC has to temporarily relocate those upper containers to fetch the lower one. This extra container movement, called *rehandling*, is thought to be the major source of inefficiency in most container terminals. Such rehandling is unavoidable when, for example, a heavier container is stacked under the lighter ones, so a good loading sequence cannot be obtained unless the containers are appropriately stacked in the yard.

Some previous studies have developed strategies to determine an appropriate stacking location for a container that is carried into the yard (Dekker et al. 2006; Ibrahimi et al. 1993; Kim et al. 2000; Preston and Kozan 2001; Yang and Kim 2006; Zhang et al. 2003). Unfortunately, rehandling cannot be completely avoided for various reasons. First, the loading sequence is often not available at the time a container is carried into the yard. The export containers typically start coming into the terminal more than 2 weeks in advance of the shipping time. However, the loading sequence for a vessel is determined only after most of the export containers for that vessel have arrived and been stacked in the yard. Obviously, it is not possible to determine optimal slots for the carried-in containers by considering a loading sequence which does not exist. Second, the information provided on an export container at the time of its arrival is imprecise. Many terminal operators use the weight information to determine appropriate slots for the export containers without knowing the exact loading sequence. For example, rehandling can be avoided if the containers belonging to the same weight class are stacked together. However, for many of the containers, the weights are estimated at the time of arrival and the precise weight information is obtained much later (Kang et al. 2006). Third, the storage capacity of the yard is limited. Therefore, there is no guarantee that the slot for an export container that appeared to be the best at the time of arrival is still optimal at the time of loading. As a worst-case example, if no ground position is available when the export container that is the last in the loading sequence enters the yard, rehandling is inevitable irrespective of where the container is stacked.

Remarshaling refers to the task of relocating the export containers into a proper arrangement for the purpose of increasing the efficiency of the loading operation. In this paper, we propose a method based on our preliminary work for planning for remarshaling assuming that the ordering constraints for loading are known (Kang et al. 2006). Although the containers are often relocated from block to block for remarshaling, for purposes of this study we only consider the case in which the containers are to be rearranged within a block. As is common in most semi-automated container terminals, we also assume that a block is equipped with two automated SCs which cannot move across each other. This requires our method to consider the problem of interference between the two SCs.

Related works

Not much work has been done on remarshaling. Lee and Hsu solved what they call a pre-marshaling problem which in our term is an intra-bay remarshaling problem (Lee and Hsu 2007). Given a loading sequence, the goal of pre-marshaling is to reshuffle the containers in each bay with the least number of movements so that no rehandling occurs during the loading operation. Since the containers are relocated only within one bay, a single SC is used without the need of considering job assignment and interference between the cranes. Kim and Hong proposed a heuristic method for relocating containers within a bay during the loading operation (Kim and Hong 2006). Their heuristic determines a good location for a rehandled container to be placed to minimize further rehandlings until all the containers within a bay are loaded. This method is useful when remarshaling cannot be carried out in advance of the loading operation, but is not intended for remarshaling itself. Kim and Bae presented a two-stage approach to planning for remarshaling (Kim and Bae 1998). In the first stage, the containers and the target bays to which they are to be moved are determined in such a way that the number of containers to be moved is minimized for a given ship stowage profile. In the second stage, the sequence of container moves is determined so that crane movements can be minimized. However, since they assumed that the loading sequence is not available, their algorithm does not determine the exact target slots of individual containers in the target bays and thus does not guarantee a complete prevention of rehandling. In contrast, our algorithm determines the storage slots of individual containers in the given target bays to prevent rehandling during the loading operation. Another difference is that Kim and Bae's work assumed a single SC in a block and so did not consider crane interferences.

The proposed method

Our method divides the remarshaling problem into two subproblems in a manner similar to Kim and Bae's algorithm: determining the target slots to which the containers are to be moved and then scheduling the SCs to actually move the containers. In determining the target storage slots of the individual containers in the given target bays, our method aims at a complete prevention of rehandling during loading as well as during remarshaling. In scheduling the SCs, our method tries to minimize interference between the SCs and builds a crane schedule wherein all target containers are moved to their target slots in as little time as possible.

When the number of containers to be moved is large, there are a huge number of stacking configurations for the containers in the target bays that guarantee a complete avoidance of rehandling during both loading and remarshaling. Even though such configurations are all rehandling-free, the time required to actually perform remarshaling differs from one configuration to another. Since the time allowed for the remarshaling operation is often limited in most real situations, it is necessary to search for a target stacking configuration that demands the least amount of time to finish remarshaling. The proposed algorithm conducts a simulated annealing search (Aarts and Korst 1997) in the space of target configurations. Simulated annealing has been widely used for

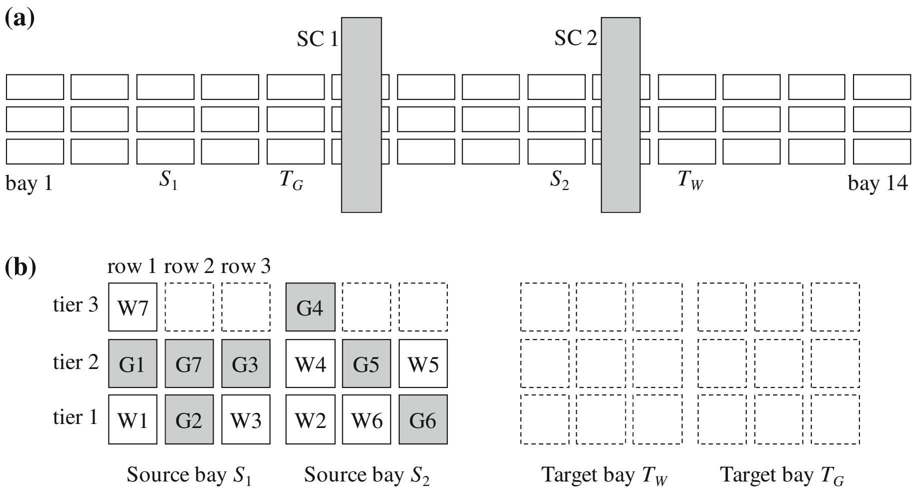

Fig. 1 An example of a remarshaling problem: **a** the top view of a block and **b** the cross sectional views of the source and target bays

planning and scheduling problems because it often finds a near-optimal solution in a reasonable time (Harhalakis et al. 1990; Palmer 1996). During the search, our algorithm evaluates a candidate target configuration by building a crane schedule and estimating (through simulation) the time taken to complete remarshaling to achieve that particular configuration. Since the crane scheduling is yet another search problem, the whole algorithm for deriving a minimum-time remarshaling plan takes the form of a nested search.

The next section introduces the remarshaling problem in greater detail. “Searching for a good target configuration” explains how we search for a good target configuration by using a simulated annealing algorithm. “Evaluation of a target configuration by crane scheduling” describes our heuristic algorithm, which evaluates a given configuration by building a crane schedule for moving the target containers. In “Experimental results”, the proposed method is tested and the results are summarized. Finally, in “Conclusions” section, conclusions are given along with some topics for further research.

Intra-block remarshaling problem

Crane operations in a block

In the stacking yard, a block consists of a number of bays each of which consists of several rows of container stacks of a certain height. Figure 1 illustrates a simplistic block with fourteen bays that are three rows wide and three tiers high. Since the handover of containers between the SCs and yard tractors takes place at the sides of the bays, the empty travel of an SC can be minimized and thus its productivity maximized during the loading operation if the containers to be loaded consecutively are stacked in the same bay as much as possible. The containers having the same port of destination with the same size are usually loaded consecutively, while the heavier ones should be loaded before the lighter ones. The containers destined for the same port having the same size are often said to belong to the same *category*. To minimize the empty travel of the SC during the loading operations, the containers of the same category should be stacked adjacently in the same bay whenever possible. To avoid rehandling during the loading, either the heavier ones must be stacked on top of the lighter ones or those belonging to the same weight class must be placed on a same stack because there is no ordering constraint among the same-weight containers.

An important goal of the intra-block remarshaling is to arrange together, in designated bays, those containers that are of the same category but are scattered around in the block. The stacking configuration of the containers in the designated bays must be carefully determined so that rehandling is avoided during the loading. It is assumed in this paper that the category of each container, the relative order of loading among the containers belonging to the same category, and the emptied bays into which the containers are to be moved have been given. The weight classes of the containers need not be considered any further because they are already taken into account when the order of loading the containers is determined. Figure 1 shows an example of an intra-block remarshaling problem. The bays in which the containers are stacked before the remarshaling are called the *source bays* and the empty bays into which the containers are to be moved are called the *target bays*. Figure 1a shows the top view of two source bays S_1 and S_2, and two target bays T_W and T_G. Figure 1b shows the cross-sectional views of the source and target bays. A rectangle drawn by solid lines in the source bays represents a container. In this example, the containers of two different categories, W and G, are residing in the source bays and these need to be moved to the

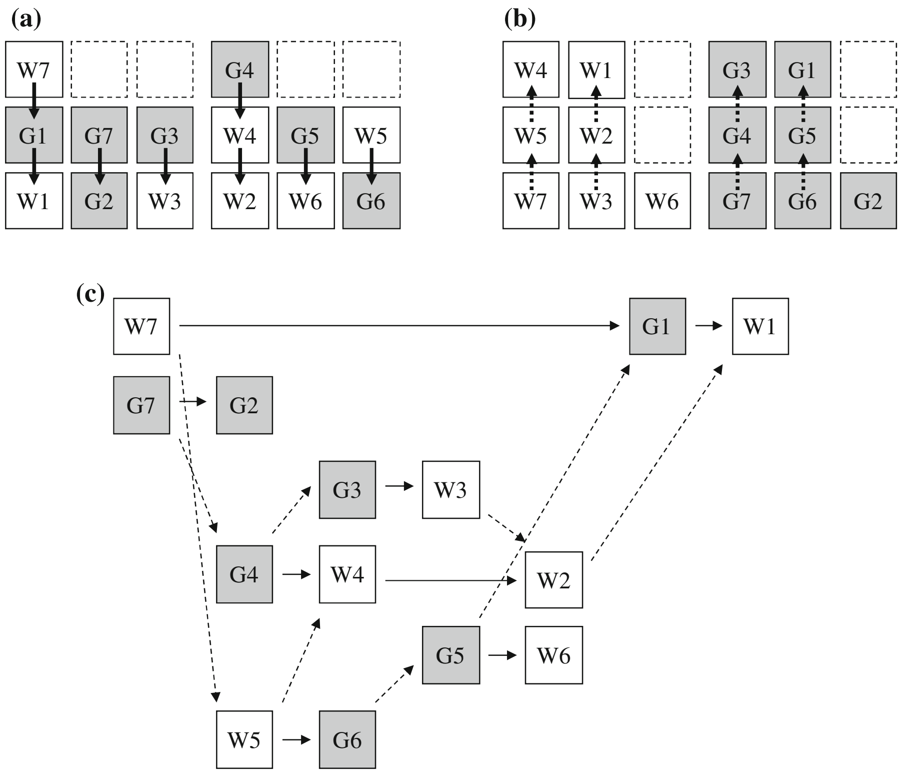

Fig. 2 Ordering constraints of the container movements: **a** the source configuration, **b** the target configuration, and **c** the partial order graph derived from the two configurations

target bays. Each container is given an alphanumeric label in which the alphabet indicates its category (and thus its target bay) and the number refers to its order in the loading sequence.

Target configuration and the partial-order graph

A plan for the intra-block remarshaling is derived by first determining the stacking slots of the target containers in the target bays and then scheduling the SCs so that the containers are efficiently moved to those slots. There are two types of constraints that must be satisfied by a rehandling-free remarshaling plan.

- Each container should be loaded onto a ship without rehandling after the remarshaling. (Type I constraint)
- Each container should be moved from its source bay to its target bay without rehandling during the remarshaling. (Type II constraint)

The configuration of the source bays restricts the order of container moves during the remarshaling due to the Type II constraint. If container c_i is placed on top of another container c_j on the same stack in the source configuration, then c_i should be moved to its target bay before c_j to avoid rehandling during the remarshaling. Similarly, the configuration of the target bays, once determined, also restricts the order of container moves during the remarshaling to satisfy the Type II constraint. If c_i is to be put on top of c_j on the same stack in the target configuration, then c_j must be moved to the target bay before c_i to avoid rehandling during the remarshaling. Notice that the target configuration itself should be determined with respect to the loading sequence in order to satisfy the Type I constraint.

Suppose a target configuration is given. Then, a partial order graph can be derived by combining the two types of ordering constraints imposed by the source and target configurations. This partial order graph encompasses all the possible rehandling-free movement sequences given the two configurations. Figure 2 shows an example of the partial order graph. Figure 2a shows the source configuration defined by the initial positions of the containers in the source bays for the remarshaling problem shown in Figs. 1 and 2b shows one possible target configuration that satisfies the Type I constraint. According to this configuration, containers belonging to the same category (W or G) are stacked in the same bay and the loading sequence is respected. The partial order graph that is derived is shown in Fig. 2c. A solid arrow represents an ordering constraint imposed by the source configuration, and a dotted arrow represents an ordering constraint imposed by the target configuration.

When determining the target configuration, it should be ensured that there is no cycle in the partial order graph derived from it. Figure 3 shows a simple example of a source and target configuration pair for which a feasible crane schedule does not exist because of a cycle in the partial order graph.

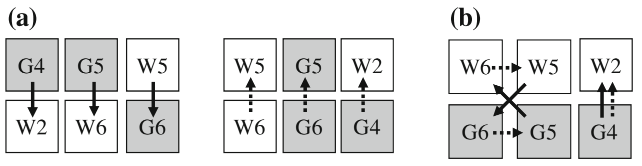

Fig. 3 An example of a partial order graph having a cycle: **a** source and target configurations and **b** the partial order graph

A cycle W6 → W5 → G6 → G5 → W6 is observed in the partial order graph of Fig. 3b that is derived from the source and target configurations of Fig. 3a. The detailed algorithm for determining the target configuration without incurring a cycle will be explained in "Searching for a good target configuration".

Crane scheduling

Given a partial order graph, we look for a crane schedule that is not only compatible with the partial order graph, but also optimal in the time required to transfer all the target containers from the source bays to the target bays. There have been some previous studies on crane scheduling for a block with multiple cranes. Kim et al. (2002) discussed crane dispatching rules for a block with two stacking cranes. Ng (2005) presented a dynamic-programming-based heuristic approach to crane scheduling for minimizing the waiting time of trucks. Dynamic programming is used to partition the block into ranges, and each crane operates exclusively in each range. In this way, these ranges help to resolve the problem of crane interference. However, these studies are not directly applicable to scheduling cranes for remarshaling because a primitive task of remarshaling is moving a container from one place to another in the block rather than loading or unloading a container from a truck. Notice that the operation of unloading from a truck never involves any rehandling. Moreover, remarshaling must respect the ordering constraints imposed by the container loading sequence.

Figure 4 shows an example of a crane schedule which satisfies all of the ordering constraints imposed by the source and target configurations shown in Fig. 2. For example, container G1 satisfies the constraint from the source configuration, i.e., G1 is moved after W7 is moved, and it also satisfies the constraint from the target configuration, i.e., G1 is moved after G5 and G6. The detailed algorithm for building a crane schedule from a given partial order graph is explained in "Evaluation of a target configuration by crane scheduling".

Overview of the proposed algorithm

As mentioned earlier in "The proposed method", our algorithm tries to find an optimal target configuration that takes the least amount of time to complete the remarshaling. We apply a simulated annealing (SA) algorithm to efficiently find a good target configuration within a reasonable amount of computational time. As shown in Fig. 5, our SA algorithm searches for a good target configuration, or equivalently, a partial order among containers to be rearranged that requires less time for remarshaling. Initially, a target configuration is generated heuristically and then the search continues by repeatedly generating neighbor configurations and evaluating them until the stopping condition is met. "Generating an initial configuration" explains how an initial target configuration is generated by using a heuristic method. "Generating neighbor configurations" gives the details of generating neighbor configurations from the current configuration. A candidate solution, which can be expressed in the form of a partial order graph, represents a set of possible container movements. A candidate solution is evaluated by heuristically constructing a full crane schedule and estimating the total amount of time needed to move all the target containers. That is, the estimated crane working time is used as the measure of quality of the target configuration.

Given a target configuration, there are also a huge number of feasible crane schedules for performing the remarshaling, although the time taken to complete the operation differs from schedule to schedule. For this reason, another search is required to find a good crane schedule that can complete the remarshaling for the given target configuration in the shortest possible time. To make this search feasible, the proposed algorithm approximates an optimal crane schedule using a heuristic algorithm. The heuristic algorithm simulates the operation of the SCs and constructs a crane schedule incrementally by repeating, from the beginning to the end of the

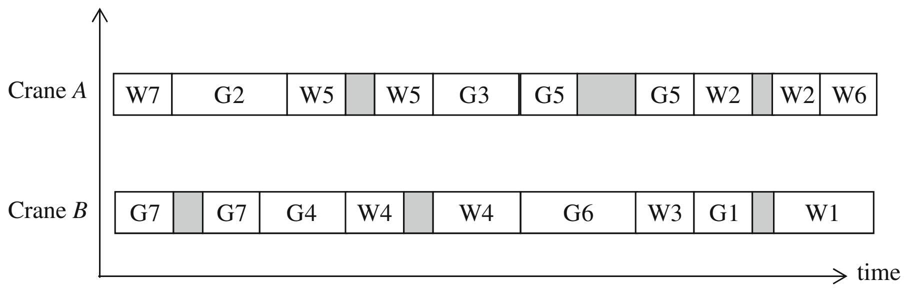

Fig. 4 An example of a crane schedule for cranes *A* and *B*

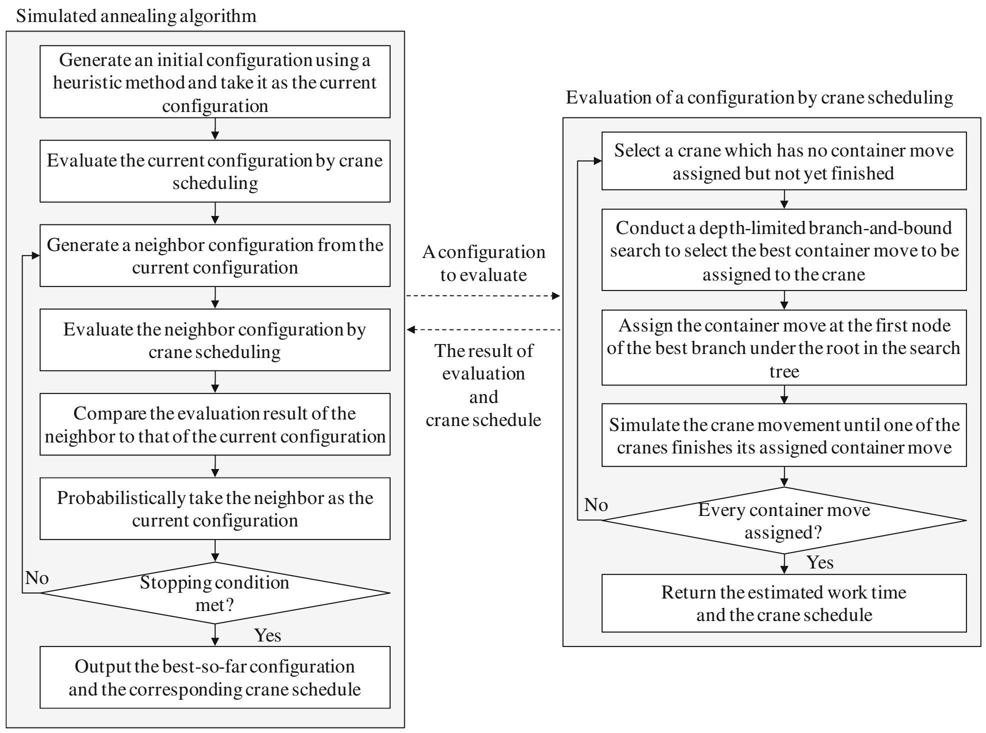
Simulated annealing algorithm
Generate an initial configuration using a heuristic method and take it as the current configuration
Evaluate the current configuration by crane scheduling
Generate a neighbor configuration from the current configuration
Evaluate the neighbor configuration by crane scheduling
Compare the evaluation result of the neighbor to that of the current configuration
Probabilistically take the neighbor as the current configuration
Stopping condition met?
No
Yes
Output the best-so-far configuration and the corresponding crane schedule
A configuration to evaluate
The result of evaluation and crane schedule
Evaluation of a configuration by crane scheduling
Select a crane which has no container move assigned but not yet finished
Conduct a depth-limited branch-and-bound search to select the best container move to be assigned to the crane
Assign the container move at the first node of the best branch under the root in the search tree
Simulate the crane movement until one of the cranes finishes its assigned container move
Every container move assigned?
No
Yes
Return the estimated work time and the crane schedule

Fig. 5 Overview of the algorithm to search for a minimum-time remarshaling plan

Table 1 Frequently used notations

Notation	Description
T_A	The target bay of the containers that belong to category A
An	The container to be loaded nth among those in category A
C_t	The set of directly movable containers satisfying the Type II constraint for the selected target bay t
nE or nL	The state of the crane; n is the bay number where the crane is located. E and L indicate the loading status of the crane; E indicates that the crane is empty and L indicates that the crane is loaded.
ET	Empty travel of the stacking crane
PU	Pick up operation of the stacking crane
LT	Loaded travel of the stacking crane
DO	Drop-off operation of the stacking crane
a_i	ith node of crane A in the state transition graph
L	The depth-limit of the depth-limited branch-and-bound search algorithm

remarshaling, the dispatching process that selects a container move to be assigned to the SC that is just released from its previous assignment. In the dispatching process, a depth-limited branch-and-bound search is conducted to find the best assignment of the SC just released. The deeper the depth, the better choice for dispatching can be made because the effect of dispatching on the successive operations of the SCs further into the future is investigated. Therefore, a larger depth limit gives a more accurate evaluation of a given configuration by building a better crane schedule. However, since the computation time increases exponentially with the depth limit, increasing the depth limit results in an exponential decrease in the number of candidate configurations evaluated by the SA search when the total amount of time for the search is limited. For this reason, finding a good balance between exploring more configurations and evaluating each configuration more in-depth is important for obtaining a good remarshaling plan. "Incremental crane scheduling by a depth-limited branch-and-bound search" gives details of the depth-limited branch-and-bound search, and "Simulation of the crane operation using state transition graphs" explains how the crane movements are simulated to estimate the time taken to move the containers. Table 1 summarizes the notations used in the sections that follow.

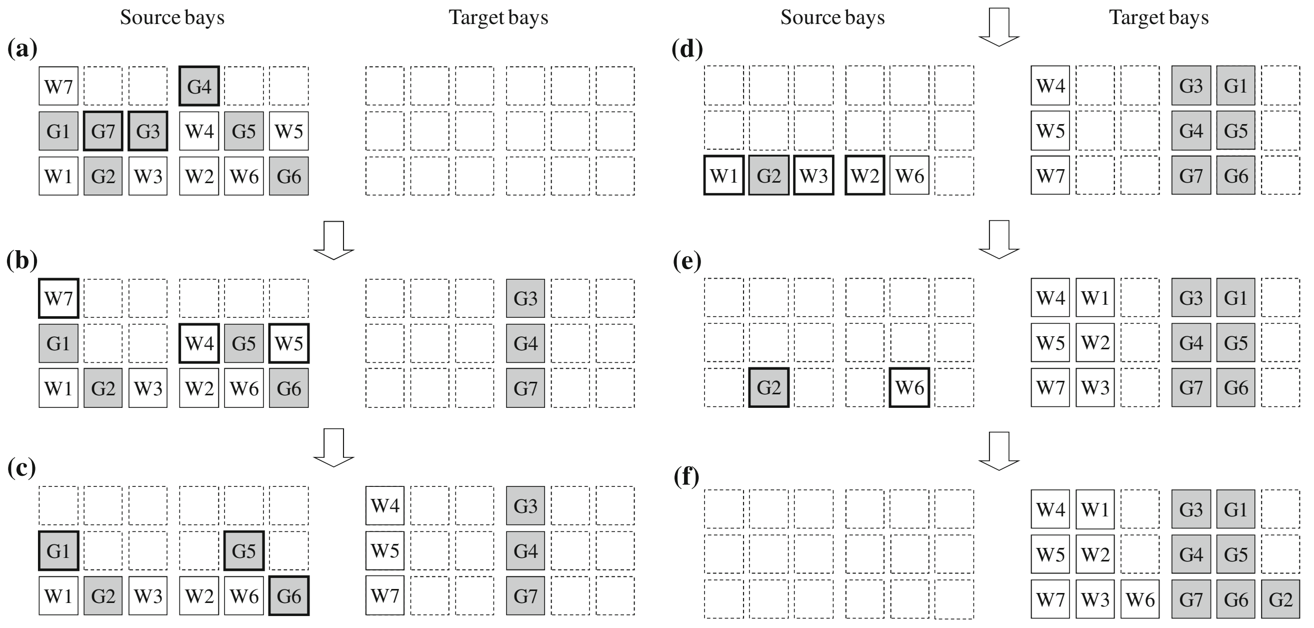

Fig. 6 Example of determining the target slots of containers

Fig. 7 Procedure for selecting the containers to fill a stack

1. For each target bay, count the number of directly movable target containers that can be moved without violating the Type II constraint.
   1.1. Select a target bay *t* that has the greatest number of directly movable target containers.
   1.2. In case of a tie, one is chosen randomly.

2. Let C_t be the set of directly movable target containers for the selected target bay *t*. From C_t, select a sufficient number of containers to fill one of the empty stacks of *t*. (In Fig. 6, a maximum of three containers can be stacked together as one stack)
   2.1. Select the container that is stacked at the highest slot in the source bay.
   2.2. In case of a tie, select the container that has below it the most number of containers to be moved to *t*.
   2.3. If a tie still exists, choose one at random.
   2.4. The selection is repeated until a sufficient number of containers are selected or there are no more directly movable target containers left.

Searching for a good target configuration

Generating an initial configuration

We use a heuristic method to generate an initial configuration of the target bays which satisfies both the Type I and Type II constraints introduced in the previous section. Our heuristic determines the configuration of the target bays one stack at a time. First, a stack in the target bays is chosen and the heuristic selects the containers to be moved to the stack considering the Type II constraint. Then, it determines the target slot of each of the selected containers on the chosen stack considering the Type I constraint. This procedure is repeated until the target slots of all the containers are determined.

Figure 6 gives a step-by-step illustration of the heuristic. In the initial state (Fig. 6a), containers W7, G7, G3, G4, G5, and W5 are located at the top positions of the stacks in their source bays. These six containers can be moved to their target bays without violating the Type II constraint. Among these containers, those to be moved to the same target bay are selected and the order is determined to move them without violating the Type I constraint. For example, G7, G4, and G3 can be selected as those to be moved to the same target bay. Their order of moves should be G7 → G4 → G3 in order to satisfy the Type I constraint. Although there may be more than one way of selecting containers for a target stack, only one movement order is possible for each selection. Figure 7 shows the detailed procedure of selecting containers for filling a target

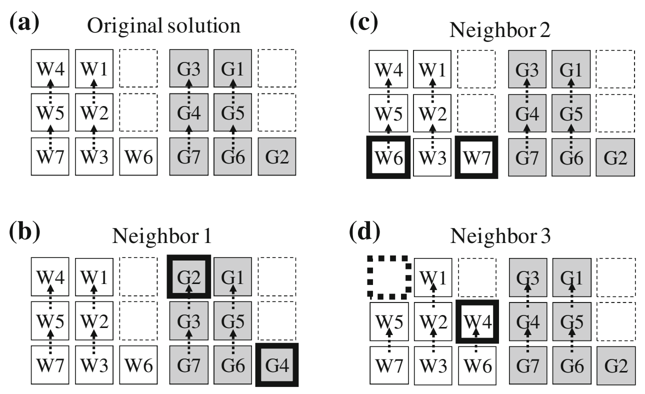

Fig. 8 Three examples of neighbor configurations generated from the solution shown in Fig. 2

stack. In Fig. 6, G3, G4, and G7 are first selected and the order of moves is G7 → G4 → G3. After removing these three containers from the source bays and placing them in a stack in the target bay, the state of the bays will be as shown in Fig. 6b. Then the next selection of containers happens to be W7, W5, and W4, whose order of moves is W7 → W5 → W4. The container selection to fill a target stack is repeated until all the containers are removed from the source bays.

For the two containers A and B stacked in the same source bay, the heuristic in Fig. 7 determines the target slot of A prior to B if an ordering constraint $A \rightarrow B$ is imposed by the source configuration. Later, when the target slot of B is determined, it can be placed either above A or in some other stack in the target bay. A conflicting ordering constraint $B \rightarrow A$ which leads to a cycle in the partial order graph cannot be added by this heuristic. Therefore, it is always possible to construct a feasible crane schedule from the target configuration generated by this heuristic.

Generating neighbor configurations

A neighbor configuration, or a neighbor solution in the search, is generated by swapping the containers in the two target slots randomly selected among those in the same target bay but not on a same stack. If the swapping causes a violation of ordering constraints imposed by the loading sequence, the problematic stack is rearranged to follow the loading sequence. Figure 8 shows examples of neighbor solutions generated by swapping. Figure 8a is the original solution. Figure 8b is a neighbor solution generated by swapping the containers G2 and G4 and rearranging the stack where G2 is newly placed. Figure 8c is a neighbor solution generated by swapping W6 and W7, which requires no rearrangement. Figure 8d is a neighbor solution generated by swapping W4 with an empty slot above W6.

The swapping operation can sometimes incur a cycle in the partial order graph, from which no crane schedule can be derived. If this happens, the neighbor solution incurring a cycle is simply abandoned and another swap is attempted. Cycle detection can be performed in an amount of time that is linear in the number of containers (Nivasch 2004). Figure 9 shows the partial order graphs derived from the three neighbor solutions of Fig. 8, where the graph of Fig. 9c contains a cycle.

Evaluation of a target configuration by crane scheduling

In this section, we explain in detail how a given target configuration is evaluated. Our evaluation algorithm first constructs a crane schedule that can achieve the target configuration. We assume that there are two SCs that cannot move across each other. Crane interference is detected by simulation using a state transition graph model. Then, the target configuration can be evaluated by estimating the total amount of time taken by the two SCs to move all the containers to complete the target configuration.

Incremental crane scheduling by a depth-limited branch-and-bound search

A crane schedule to achieve a target configuration is built incrementally by using simulation. When a crane finishes an assigned container move, the next possible container move is heuristically selected and assigned to the crane. The heuristic examines all the possible assignments and selects the best one to incrementally add up on the partially built crane schedule. A container move is considered as a possible candidate for an assignment if it is not yet assigned to any crane and does not violate any ordering constraints imposed by the partial order graph derived from the target configuration. Figure 10 shows a snapshot of incrementally building a crane schedule using our heuristic. In this figure, many container moves are already assigned and crane A is currently moving container W2. Crane B has just finished moving container W3 and thus needs a new container assignment. Under the constraints of the given partial order graph, G1 and W6 are two possible containers that can be assigned to crane B. To see which choice is better, we search through some further steps, say L steps, into the future considering all the possible subsequent container assignments, and estimate the time taken in each branch of the search to move the containers during those L steps. Then, a choice is made that leads to the best branch with the minimum estimated time. The greater the number of future steps investigated, the more likely it is that the choice made will be the best one available. However, only a small L is feasible in practice because the number of search nodes increases exponentially with L. Instead, this heuristic search is carried out every time a crane is released from its previous job. A choice for the container assignment for the released

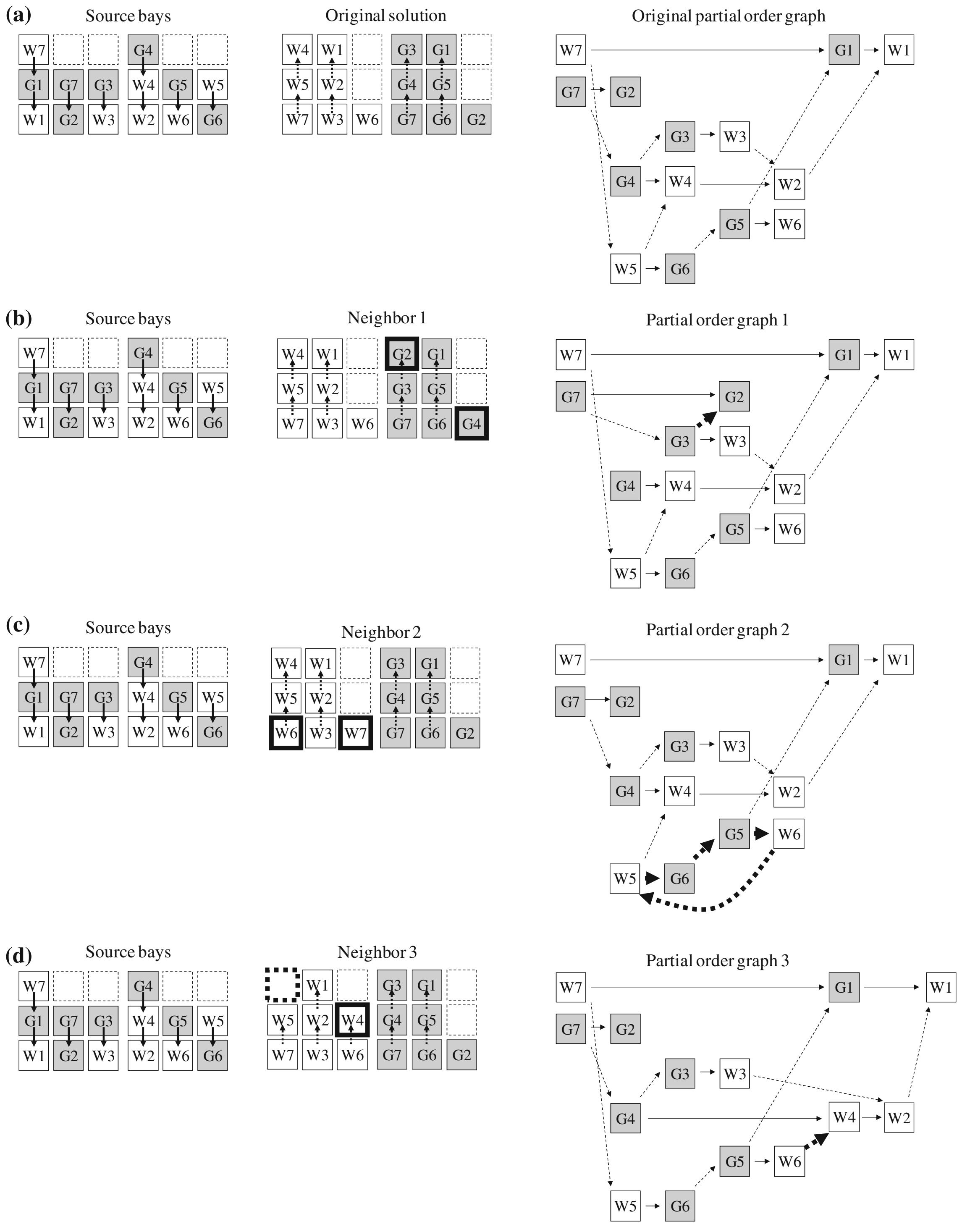

Fig. 9 Partial order graphs of the neighbor solutions of Fig. 8

crane is always made by looking into the next *L* future steps from that point in time.

The heuristic search algorithm that we use is a depth-limited branch-and-bound algorithm (or a depth-limited A* without using any heuristic function) (Land and Doig 1960). Since the search is initiated every time one of the cranes is released from its previous job, the root node represents the container assignment to the crane that is still working and the states of both cranes describing their locations and loading statuses. Each of the other nodes in the search tree represents the crane priority in addition to the container assignments chosen for both cranes and the states of the two cranes. Since

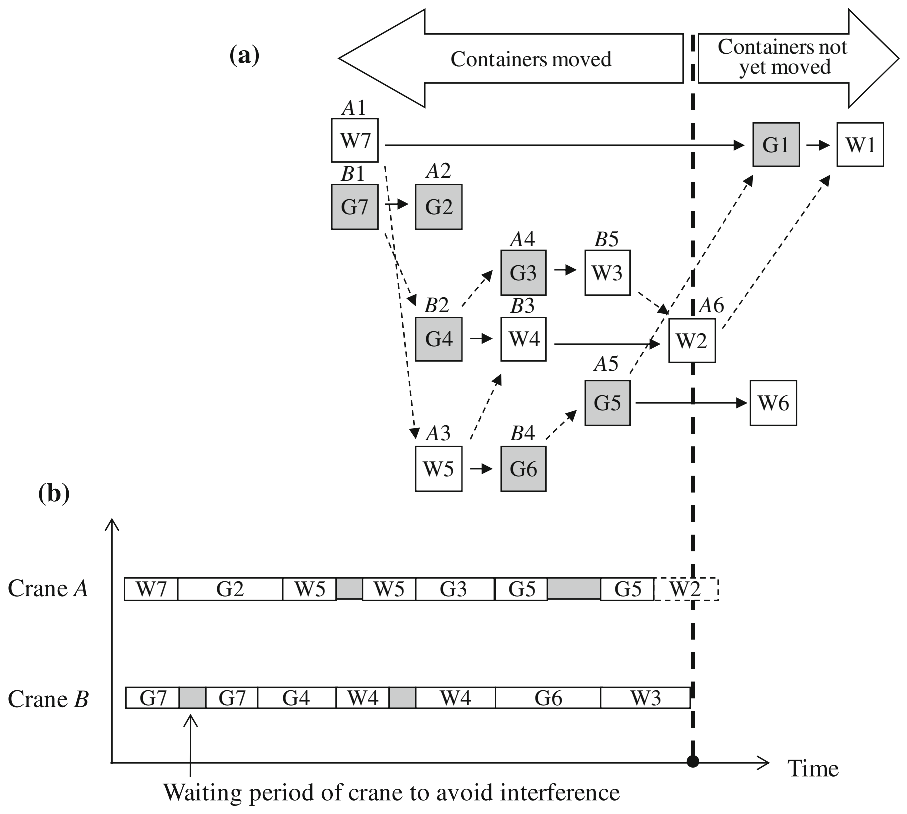

Fig. 10 An example of a partially built crane schedule: **a** a given partial order graph and **b** the partially built crane schedule

there are two cranes working that can interfere with each other, we resolve the interference by prioritizing the operation of the cranes. While the crane with priority can proceed without interruption, the other crane must wait at the nearest location outside the clearance range of the crane with priority. A node is expanded during the simulation whenever one of the cranes finishes its previous assignment. At this time, child nodes are generated for all combinations of possible container assignments to the crane just released, and for the priority assignment to the cranes.

A branch under the root node of the search tree represents a series of possible container movements by the two cranes. A branch is cut off if it reaches the predetermined depth limit. After the search is finished, we can find the best container to be assigned to our target crane from the first node of the branch under the root with the minimum estimated crane work time. Every node in the search tree is evaluated by the total crane work time so far. The total crane work time of a node is calculated by summing the total crane work time of its parent node and the work time spent while moving the container assigned at that node. The details of the crane operation simulation and the calculation of the work time including the delay caused by any interference are explained in the following sub-section. The branch-and-bound algorithm saves search time by examining the evaluation value of each node to prune unpromising branches and by exploring the nodes with the best-first search strategy.

Figure 11 shows a search tree generated by the branch-and-bound algorithm with a maximum depth of four. The prioritized crane in each node is indicated by a bold-faced underlined letter. The number at the right side of each node denotes the total crane work time. The rectangle with the slant lines indicates that no container is assigned to that crane. In this figure, the branch-and-bound algorithm expands the nodes in the order $n_0 \rightarrow n_1 \rightarrow n_5 \rightarrow n_6 \rightarrow n_8 \rightarrow n_{10} \rightarrow n_3 \rightarrow n_7 \rightarrow n_4 \rightarrow n_2 \rightarrow n_9$. The best branch under the root is $n_1 \rightarrow n_6 \rightarrow n_8 \rightarrow n_{10}$. Container G1, the first assignment of the best branch, is selected for a new container assignment to crane B and priority is given to crane A which is moving W2. The partially built crane schedule shown in Fig. 10 is expanded by adding the selected container G1 to crane B. Then, another depth-limited branch-and-bound search will be performed to search for the next new container assignment for crane A when it is released from its work of moving container W2. This process is repeated until all the containers are assigned to one of the cranes.

The maximum depth L of the depth-limited search can be controlled. A better overall crane schedule can be obtained with a larger L value. If the computation time allowed for the search is limited, however, a larger L results in fewer trials in the search for a good target configuration because it takes more time to evaluate each configuration. Experimental results have shown that a better remarshaling plan can be obtained by having a good balance between exploring more

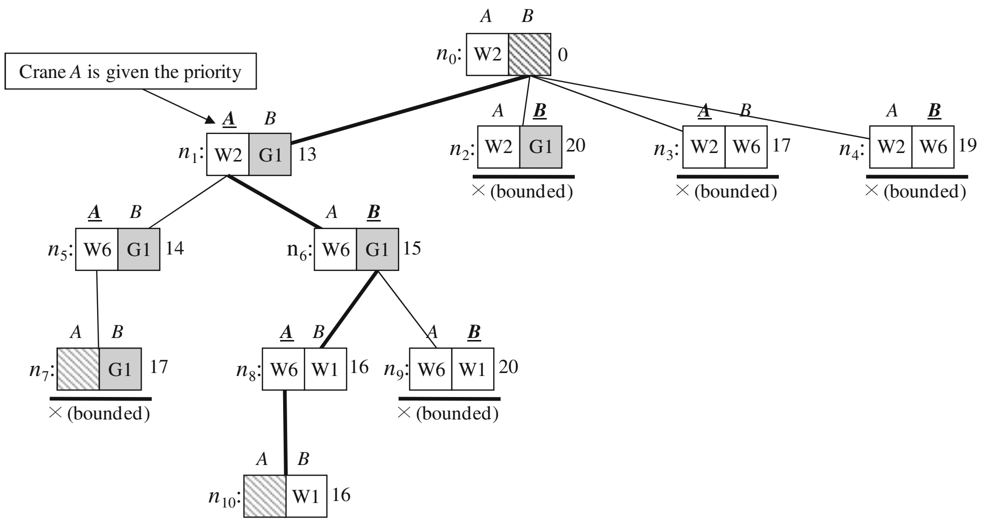

Fig. 11 The search tree explored by the depth-limited branch-and-bound algorithm in order to find a container assignment for crane *B*

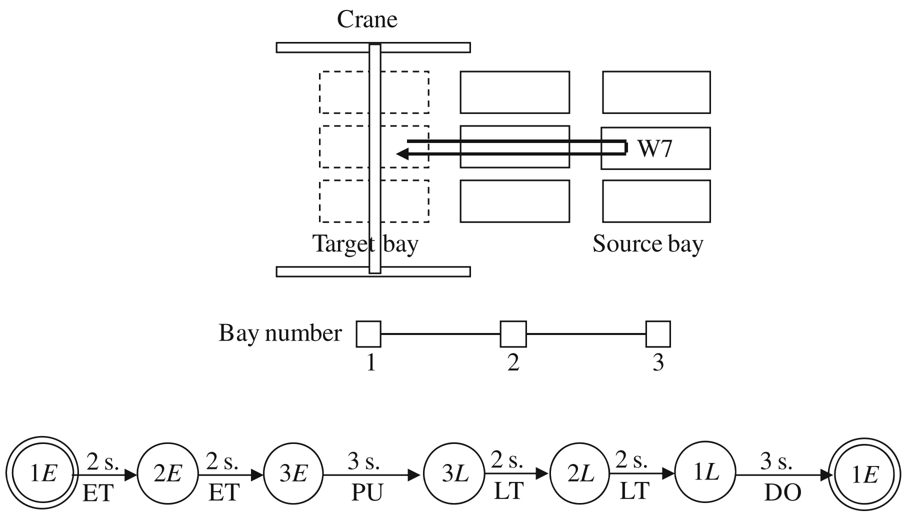

Fig. 12 A state transition graph of a crane that is attempting to move container W7 from bay 3 to bay 1

target configurations and evaluating each configuration more deeply.

Simulation of the crane operation using state transition graphs

To evaluate each node during the branch-and-bound search explained above, we need methods for both detecting the crane interferences and calculating the waiting times required to avoid them. For these, the location of each crane at any specific moment is predicted by simulating the crane movement using a state transition graph. A node in the state transition graph represents the state of a crane and a directed edge represents a state transition. Figure 12 shows an example of a state transition graph which represents a crane moving container W7 from bay 3 to bay 1. The number in each node refers to the bay number at which the crane is located. The alphabet indicates the loading status of the crane: *E* implies that the crane is empty and *L* implies that the crane is loaded, i.e., carrying a container. For example, 3*E* indicates that the crane is currently empty and located at bay 3.

The state transition is made by any one of the four different crane operations needed for remarshaling: empty travel to the source bay (ET), pickup of the target container from the source bay (PU), loaded travel to the target bay (LT), and dropping-off the target container at the target bay (DO). The operations ET and LT change the location of the crane, while PU and DO only change the loading status. For each directed edge, the crane operation making the transition is marked below the arrow and the time required for the operation is written above the arrow. The state of the crane at a specific time can be easily predicted by following the arrows in sequence and by summing the transition times. According to the example shown in Fig. 12, the crane remains at the first state for 2 s and at the second state for another 2 s. Starting from bay 1, it takes 14 s to pick up W7 and stack it at the target bay.

When two cranes cannot move across each other, interference between them can be detected by constructing the state transition graph for each crane and comparing them. Figure 13 shows an example of constructing the state transition graphs to detect interferences between cranes A and B when container G1 is assigned to crane B while crane A is working for container W2. It is assumed that interference occurs if the distance between the two is less than two bays. For each of the two cranes, the state transition graph can be easily built without considering the other's movements. Then, the pair of nodes in danger of being involved in the interference can be detected by comparing the bay numbers and the arrival times at the nodes in the two state transition graphs. Let L_A and L_B be the bay locations of cranes A and B, respectively. A necessary condition for interference is $|L_A - L_B| < 2$. In Fig. 13, the four node pairs connected by dashed lines are potentially involved in the interference. Interference occurs if the two cranes reach any of these node pairs simultaneously.

Interference is easily detected by simulating the movement of cranes following the state transition graphs. The examples in Fig. 14a, b show the states when 2 and 4 s have elapsed from $t_0$for the example shown in Fig. 13, respectively. Let the ith node of crane A be a_i, and the jth node of crane B be b_j. Assume that the two cranes start their work simultaneously at t_0. The initial states of cranes A and B are a_1 and b_1, respectively. Two seconds later, the state of A is a_2 and that of B is b_2, as shown in Fig. 14a. The state of each crane at a specific moment is indicated by a thick arrow in the figure. After another 2 s, the state of B changes to b_3 while that of A remains at a_2. Since the nodes a_2 and b_3 are one of the pairs previously recorded for potential interference, the interference is unavoidable if the two cranes keep moving without interruption. To avoid this interference, the crane without priority must remain and wait at its previous state until the crane with priority changes its state. In Fig. 14b, assuming that crane A has priority, crane B waits at b_2 and checks if it can move to the next node b_3 every time the state of A changes. After 5 s of delay at b_2, which is 9 s after starting at t_0, crane B can safely move to b_3. Figure 14c shows the states of the cranes at that moment.

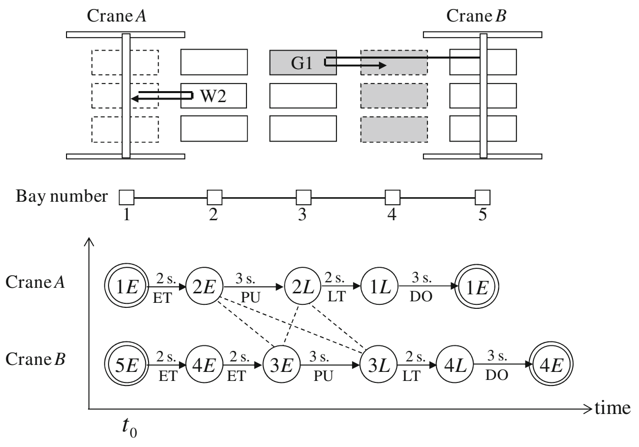

Fig. 13 State transition graphs for detecting potential interference

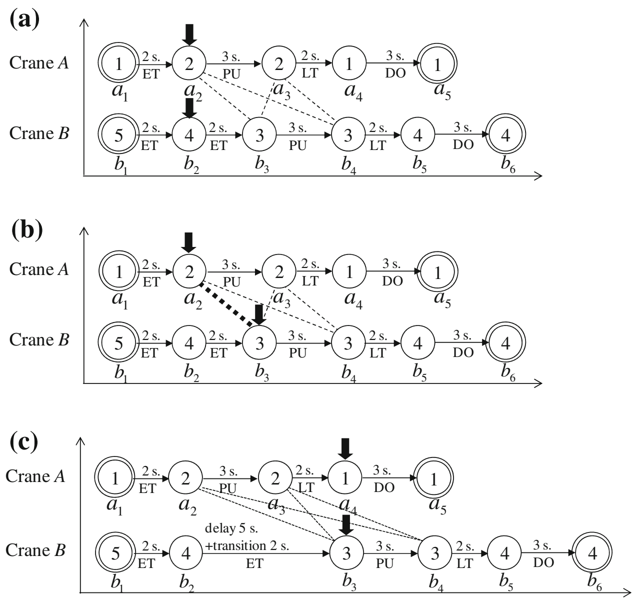

Fig. 14 Interference detection using the state transition graphs: **a** at $t_0 + 2$s, **b** at $t_0 + 4$s, and **c** at $t_0 + 9$s

Experimental results

A block is assumed to have 33 bays, each of which is nine rows wide and six tiers high. It is also assumed that two non-crossable SCs are working on the remarshaling task and they require at least a five-bay gap as a suitable working clearance. There are four empty target bays that can accommodate 196 target containers. The experiment covers three different scenarios that are different in their locations of the target bays. Figure 15 shows the location setting of the target bays in each scenario. For each scenario, simulation experiments for moving 196 containers to the target bays have been conducted with varying number of source bays (4, 6, 8, and 16). The locations of the source bays are determined randomly. When there are a larger number of source bays, the initial locations of the target containers are more widely spread around the block.

In addition to our SA algorithm, some other methods have been devised and implemented for comparison purposes. These other methods are named HRonly, HR10000, and HC. The HRonly method does not use any search algorithms; it simply determines the target configuration by using the heuristic explained in “Generating an initial configuration”, and then it constructs a crane schedule by performing a depth-limited branch-and-bound search with the depth limit $L = 1$. The performance of HRonly is used as a baseline for the

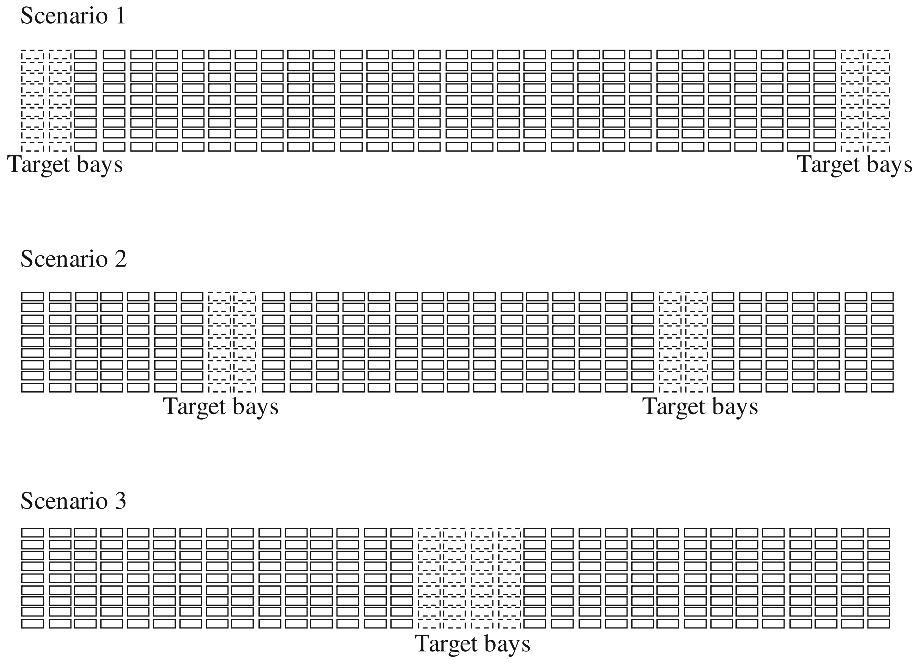

Fig. 15 Locations of the target bays in each of the experimental scenarios

Table 2 Time to complete the remarshaling tasks (min)

Location of target bays	Algorithms	Number of source bays			
		4	6	8	16
Scenario 1	HRonly	271.8	357.2	426.9	585.0
	HR10000	261.6	324.2	373.3	519.1
	HC	260.9	324.5	375.7	519.2
	SA	259.2	318.7	365.4	496.5
Scenario 2	HRonly	268.8	332.4	371.8	454.8
	HR10000	248.5	288.0	317.9	384.4
	HC	248.1	294.0	324.7	390.8
	SA	241.0	281.5	314.7	380.8
Scenario 3	HRonly	251.2	302.0	331.9	375.5
	HR10000	229.1	268.7	296.9	347.0
	HC	228.4	269.7	299.9	350.9
	SA	223.7	265.3	292.7	346.7

comparisons. The HR10000 method differs from HRonly in that it repeatedly generates 10,000 target configurations heuristically and selects the best one. It also uses the depth-limited branch-and-bound search with $L = 1$ to build a crane schedule for evaluating a configuration. The third method uses hill-climbing (HC) to search for a good target configuration, while our method uses simulated annealing (SA). The number of evaluations is limited to 10,000 for both search methods for fair comparisons. The annealing schedule for SA is appropriately adjusted as the number of source bays is changed. Evaluating 10,000 configurations by using the depth-limited branch-and-bound algorithm with $L = 1$ takes approximately 6 min on a Pentium PC with a 2.7 GHz CPU. Given this same amount of time, the depth-limited branch-and-bound algorithm can be run up to about $L = 9$ to evaluate a single candidate configuration.

Each experiment has been repeated ten times and the results averaged. Table 2 shows the estimated times needed to finish the given remarshaling tasks. Table 3 shows the percentage performance improvement of each algorithm as compared to HRonly. For all the scenarios, SA showed the best performance against all the other algorithms. Note that the performance improvement varies with the scenarios and the number of source bays. When there are many source bays, the target containers are widely distributed around the block. If the target bays are located in the middle of the block, as in the case of Scenario 3, the two SCs can move these containers almost without interference; one crane works for the

Table 3 Performance improvement relative to HRonly (%)

Location of target bays	Algorithms	Number of source bays			
		4	6	8	16
Scenario 1	HR10000	3.8	9.2	12.6	11.3
	HC	4.0	9.2	12.0	11.3
	SA	4.6	10.8	14.4	15.1
Scenario 2	HR10000	7.5	13.3	14.5	15.5
	HC	7.7	11.5	12.7	14.1
	SA	10.4	15.3	15.4	16.3
Scenario 3	HR10000	8.8	11.0	10.6	7.6
	HC	9.1	10.7	9.7	6.5
	SA	11.0	12.1	11.8	7.7

containers to the left area of the target bays and the other for those to the right. Therefore, the amount of time needed to finish the task is relatively short, and the potential for performance improvement is smaller than the potential in Scenarios 1 and 2. It can be seen from Table 3 that the actual performance improvement obtained by SA is the least with Scenario 3 when the number of source bays is 16. When the number of source bays is small, the target containers are concentrated in some specific areas in the block. In this case, it is easier for the SCs to work if the target bays are located at both ends of the block, as in Scenario 1, because they can carry out the remarshaling work with minimum interference. Therefore, the crane delay is the smallest with Scenario 1 when the number of source bays is 4, and accordingly the obtained performance improvement is the least.

Figure 16 shows the performance improvement due to the increasing predictive power by increasing L for the branch-and-bound search while limiting the iteration count of the SA search to 10,000. In the figure, a, b, c, and d show the results for 4, 6, 8, and 16 source bays, respectively. The horizontal axis is the depth limit L and the vertical axis is the percentage of the estimated remarshaling time compared to the time when L is 1. The results show that the estimated remarshaling time tends to decrease as L increases. The performance improvement also increases as the number of source bays increases because there are more options to try during the branch-and-bound search.

Figure 17 shows the execution times of the branch-and-bound search for the crane scheduling as the depth limit increases. The times are obtained by averaging the execution times of all the scenarios and it can be seen that the execution times increase exponentially as the depth limit increases. If $L > 10$, it is not feasible to apply to real-time situations because more than one hour is taken to evaluate just one candidate target configuration.

If the time available for the search is limited, there is a trade-off between the number of candidate target configurations to be evaluated and the predictive power for better crane scheduling. For the experiments, nine different pairs of iteration number I and depth limit L have been chosen that use almost the same amount of search time. These pairs have been tested under each scenario by varying the number of source bays, and the results are given in Table 4 where each entry is the remarshaling time as a percentage relative to the time taken with (10,000, 1). It can be seen that the pair exhibiting the best performance differs depending on the number of source bays. The pairs with relatively large I and small L exhibit the best performance when the number of source bays is small. When the target containers to be moved are initially concentrated in some narrow area, there will be many Type II ordering constraints imposed on the containers and there will be only limited freedom to change the order of container movements, so there will be little room for improving the crane scheduling. Instead, finding a better target configuration becomes more important because there may be chances of finding crane schedules involving less interferences by having Type I constraints imposed differently. On the other hand, the pairs with relatively small I and large L exhibit the best performance when the number of source bays is large. Since the target containers are widely scattered around the block, there will not be many Type II constraints among the target containers. This means that there is a lot of freedom to build the crane schedule in many different ways. Given any target configuration, there may be good chances to get a good crane schedule involving minimum crane interferences. Therefore, it is more important to derive a better crane schedule by having more predictive power (bigger L) than to investigate a large number of target configurations.

Conclusions

In this paper, we presented a method of searching for a good intra-block remarshaling plan assuming that the ordering constraints for the loading operation are known and the block

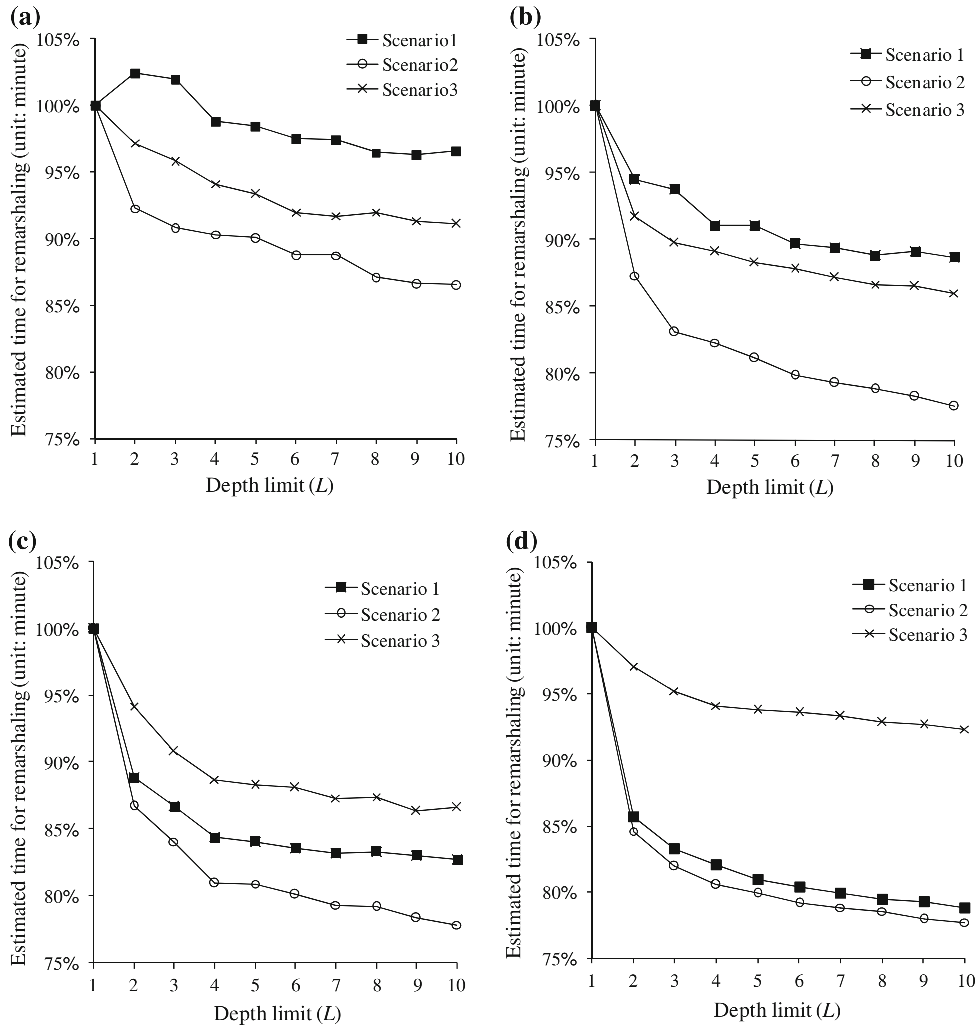

Fig. 16 Performance improvement due to the increasing predictive power, with different number of source bays: **a** 4, **b** 6, **c** 8, and **d** 16 source bays

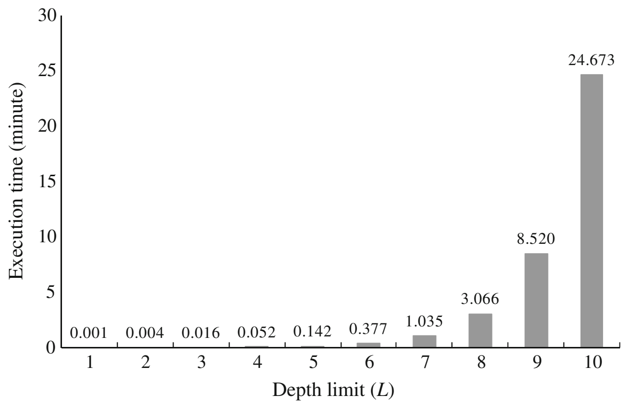

Fig. 17 Execution time of the branch-and-bound search with increasing L

is equipped with two cranes that cannot move across each other. A good remarshaling plan, when executed, should be able to efficiently rearrange all the target containers into a certain target configuration which guarantees that no rehandling will occur at the time of loading those containers onto a vessel. We have proposed to use a simulated annealing algorithm to search in the space of legitimate target configurations, evaluating each target configuration by building a crane schedule and estimating the time taken to achieve the configuration when the schedule is executed. A configuration is legitimate if it guarantees no rehandling at the time of loading. We have introduced a heuristic procedure that can easily generate a legitimate target configuration from a given initial configuration of target containers. We have also given a method that can generate a slightly different, yet legitimate, neighbor target configuration from a given legitimate target configuration.

For an evaluation of a target configuration, we first derive a partial order graph of the target containers in which the ordering constraints come from the target configuration as well as the initial configuration, and then we incrementally build a crane schedule satisfying these ordering constraints by simulating the crane operations using the state transition graphs. We have proposed to use a depth-limited

Table 4 Performance comparison of remarshaling plans obtained from nine different iteration-depth limit pairs (%)

Target bay	Source bay	Iteration number and depth-limit pairs								
		(10000,1)	(3000,2)	(1000,3)	(300,4)	(100,5)	(30,6)	(10,7)	(3,8)	(1,9)
S1	4	100.0	99.9	99.8	**99.7**	100.0	99.8	100.3	100.8	101.0
	6	100.0	98.2	98.1	**98.1**	98.2	98.6	98.6	98.8	99.3
	8	100.0	96.1	96.0	95.8	**95.5**	95.7	95.7	96.2	96.2
	16	100.0	92.2	92.0	91.5	91.8	**91.4**	91.5	91.5	91.8
S2	4	100.0	94.9	**94.7**	95.6	96.1	96.1	96.0	95.8	96.0
	6	100.0	94.3	92.5	92.3	**92.1**	92.3	92.3	92.5	92.8
	8	100.0	93.2	92.0	91.4	91.6	**91.4**	91.8	91.8	92.1
	16	100.0	95.8	94.4	94.1	93.9	**93.5**	93.8	93.9	94.3
S3	4	100.0	**99.4**	99.5	99.6	100.1	99.7	100.5	100.7	101.0
	6	100.0	98.6	98.0	97.7	**97.5**	98.0	97.9	98.4	98.9
	8	100.0	98.0	97.8	97.5	**97.2**	97.5	97.7	97.8	97.8
	16	100.0	99.7	99.4	99.0	**98.8**	99.0	99.3	99.4	100.1

Each bold-faced value indicates the best performance in each row

branch-and-bound search to choose a container to be assigned to a crane during the process of incrementally building a crane schedule. The further into the future we conduct the branch-and-bound search, the more accurate evaluation of a given configuration we obtain by being able to build a better crane schedule. However, given a limited amount of time for generating a remarshaling plan, we cannot evaluate enough number of different target configurations if the depth limit is too large. There is a tradeoff between the depth limit and the number of target configurations evaluated.

Our proposed method can generate an efficient remarshaling plan in a reasonable computational time. Experimental results have shown that a better remarshaling plan can be obtained by carefully adjusting the depth limit of the branch-and-bound search given a limited computation time. When the target containers are initially stacked densely in some narrow area, many ordering constraints exist among the target containers because many of them are on the same stack. In such cases, the crane schedules may also be tightly constrained and thus it becomes difficult to avoid interferences by trying to find a good crane schedule by increasing only the depth limit. Instead, we should investigate many different target configurations to increase the chance of avoiding the interferences. On the other hand, when the target containers are widely scattered around the block, we might be able to find a good crane schedule involving little interference given any target configuration, so we need to increase the depth limit. One of our future works would be to develop a method for a dynamic and automatic adjustment of the depth limit during the search. We are also interested in finding a smarter way of generating the neighbor configurations when selecting the containers to be swapped by considering the feedback information that can be obtained during the process of the crane scheduling.

Acknowledgements This work was supported by the Grant of the Korean Ministry of Education, Science and Technology (The Regional Core Research Program/Institute of Logistics Information Technology).

References

Aarts, E., & Korst, J. (1997). *Simulated annealing, local search in combinatorial optimization* (pp. 91–120). New York: Wiley.

Dekker, R., Voogd, P., & Asperen, E. (2006). Advanced methods for container stacking. *OR Spectrum, 28*, 563–586.

Harhalakis, G., Proth, J. M., & Xie, X. L. (1990). Manufacturing cell design using simulated annealing: An industrial application. *Journal of Intelligent Manufacturing, 1*(3), 185–191.

Ibrahimi, M. T., De Castilho, B., & Daganzo, C. F. (1993). Storage space vs. handling work in container terminals. *Transportation Research B, 27*, 13–32.

Kang, J., Oh, M. S., Ahn, E. Y., & Ryu, K. R. (2006). Planning for intra-block remarshalling in a container terminal. In *Proceedings of the 19th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2006)* (pp. 1211–1220).

Kang, J., Ryu, K. R., & Kim, K. H. (2006). Generating a slot assignment rule for outbound containers having imprecise weight information. *Journal of Intelligent Manufacturing, 17*, 399–410.

Kim, K. H., & Bae, J. W. (1998). Re-marshaling export containers in port container terminals. *Computer and Industrial Engineering, 35*, 655–658.

Kim, K. H., & Hong, G. (2006). A heuristic rule for relocating blocks. *Computers and Operational Research, 33*, 940–954.

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.

Kim, K. H., Wang, S. J., Park, Y. M., Yang, C. H., & Bae, J. W. (2002). A simulation study on operation rules for automated container yards. In *Presentation/Proceedings of the 7th Annual International Conference on Industrial Engineering*.

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica, 28*, 497–520.

Lee, Y., & Hsu, N. Y. (2007). An optimization model for the container pre-marshalling problem. *Computers and Operations Research, 34*, 3295–3313.

Ng, W. C. (2005). Crane scheduling in container yards with inter-crane interference. *European Journal of Operational Research, 164*, 64–78.

Nivasch, G. (2004). Cycle detection using a stack. *Information Processing Letters, 90*, 135–140.

Palmer, G. J. (1996). A simulated annealing approach to integrated production scheduling. *Journal of Intelligent Manufacturing, 7*(3), 163-176.

Preston, P., & Kozan, E. (2001). An approach to determine storage locations of containers at seaport terminals. *Computers & Operations Research, 28*, 983–995.

Yang, J. H., & Kim, K. H. (2006). A grouped storage method for minimizing relocations in block stacking systems. *Journal of Intelligent Manufacturing, 17*(4), 453–463.

Zhang, C., Liu, J., Wan, Y., Murty, K. G., & Linn, R. J. (2003). Storage space allocation in container terminals. *Transportation Research B, 37*, 883–903.