Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 156 (2004) 752–768

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

O.R. Applications

# A crane scheduling method for port container terminals

Kap Hwan Kim [*], Young-Man Park [1]

*Department of Industrial Engineering, Pusan National University, Changjeon-dong, Kumjeong-ku, Pusan 609-735, South Korea*

Received 22 January 2001; accepted 21 January 2003

**Abstract**

This paper discusses the problem of scheduling quay cranes (QCs), the most important equipment in port terminals. A mixed-integer programming model, which considers various constraints related to the operation of QCs, was formulated. This study proposes a branch and bound (B & B) method to obtain the optimal solution of the QC scheduling problem and a heuristic search algorithm, called greedy randomized adaptive search procedure (GRASP), to overcome the computational difficulty of the B & B method. The performance of GRASP is compared with that of the B & B method.
© 2003 Published by Elsevier B.V.

*Keywords:* Scheduling; Branch and bound; Transportation; Container terminal; Combinatorial optimization

## 1. Introduction

The productivity of a container terminal can be measured in terms of the productivity of two types of operations. One type is ship operations, in which containers are discharged from and loaded onto a ship. The other is receiving and delivery operations, in which containers are transferred to and from outside trucks. The planning process of ship operations consists of berth planning, quay crane (QC) scheduling (in practice, called work scheduling), and discharge and load sequencing. During the process of berth planning, the berthing time and the berthing position of a containership on a wharf are determined. A QC schedule specifies the service sequence of bays in a ship by each QC and the time schedule for the services. Input data for QC scheduling consists of a stowage plan of a ship, the ready time of each QC, and a yard map that shows the storage locations of containers bound for the ship. Finally, during discharge and load sequencing, the discharge and load sequence of individual containers are determined based on a QC schedule. This paper addresses the QC scheduling problem, which is pertinent to the second stage of ship operation planning.

[*] Corresponding author. Tel.: +82-51-510-2419; fax: +82-51-512-7603.
*E-mail addresses:* kapkim@pusan.ac.kr (K.H. Kim), ymanpark@pusan.ac.kr (Y.-M. Park).
[1] Tel.: +82-51-510-1483; fax: +82-51-512-7603.

0377-2217/$ - see front matter © 2003 Published by Elsevier B.V.
doi:10.1016/S0377-2217(03)00133-4

Several studies have been conducted to improve the efficiency of ship operations in port container terminals. The subject of these studies includes the following: berth planning (Brown et al., 1995; Lim, 1998), in which berthing positions and berthing times of vessel are determined; load sequencing (Cho, 1982; Cojeen and Dyke, 1976; Gifford, 1981), in which the loading sequence of individual containers are determined; and transtainer routing (Kim and Kim, 1999; Narasimhan and Palekar, 2002), in which the travel route of each transtainer and the number of outbound containers to pick up at each yard-bay are determined. In addition, Cheung et al. (2002) and Zhang et al. (2002) proposed methods for deploying yard cranes, methods in which the times and routes of yard crane movements among blocks are determined.

Daganzo (1989) was the first who discussed the QC scheduling problem. He suggested an algorithm for determining the number of cranes to assign to ship-bays of multiple vessels. Peterkofsky and Daganzo (1990) also provided an algorithm for determining the departure times of multiple vessels and the number of cranes to assign to individual holds of vessels at a specific time segment. They also attempted to minimize the delay costs. The studies by Daganzo (1989) and Peterkofsky and Daganzo (1990) assumed one task per ship-bay, a task which needs crane operations during a specific length of time, and did not consider the interference among QCs or precedence relationships among tasks. In contrast, this study assumes that there may be multiple tasks involved in a ship-bay, and thus this study divides a task into smaller sizes, compared to Daganzo (1989) and Peterkofsky and Daganzo (1990).

This study further assumes that the berthing and departure times of a vessel and the operation starting times of QCs assigned to that vessel are given. This study attempts to determine the schedule of each QC assigned to a vessel, with the goal of completing all of the ship operations of a vessel as rapidly as possible.

The next section introduces the ship operation and defines the QC scheduling problem. Section 3 provides a mathematical formulation for the QC scheduling problem. Section 4 proposes a branch and bound (B & B) algorithm for solving the mathematical formulation. Section 5 applies the heuristic algorithm GRASP to the QC scheduling problem. Section 6 discusses the results of a numerical experiment, and Section 7 provides a conclusion.

## 2. Problem description

The goal of studying the QC scheduling problem is to determine the sequence of discharging and loading operations that a QC will perform so that the completion time of a ship operation is minimized. Fig. 1 provides a drawing of QCs working on a ship.

Inbound containers with the same loading port, of the same size, and transported by the same ship are said to be included in the same container group. Likewise, outbound containers with the same destination port, of the same size, and to be loaded onto the same ship are said to be in the same container group. For sake of efficiency in the discharging and loading operations, in the stowage plan for a ship, a collection of slots that are located adjacent to each other in the ship are usually allocated to containers of the same group. In the stowage plan for loading, a cluster is defined to be a collection of adjacent slots into which containers of the same group are planned to be loaded. In the stowage plan for discharging, a cluster implies a collection of adjacent slots in which inbound containers of the same group are stacked.

Fig. 2 illustrates a stowage plan for a ship. This plan consists of four cross-sectional views, each corresponding to a ship-bay and labeled with an odd number from 1 to 7. Each small square represents a slot. Shaded squares correspond to slots that containers must be discharged from or loaded onto in this container terminals. The shaded pattern in each slot represents a specific group of containers to be loaded into or picked up from the corresponding slots. Fig. 2 shows that four groups of containers should be discharged from five clusters of slots and then four groups of containers should be loaded into five clusters of slots.

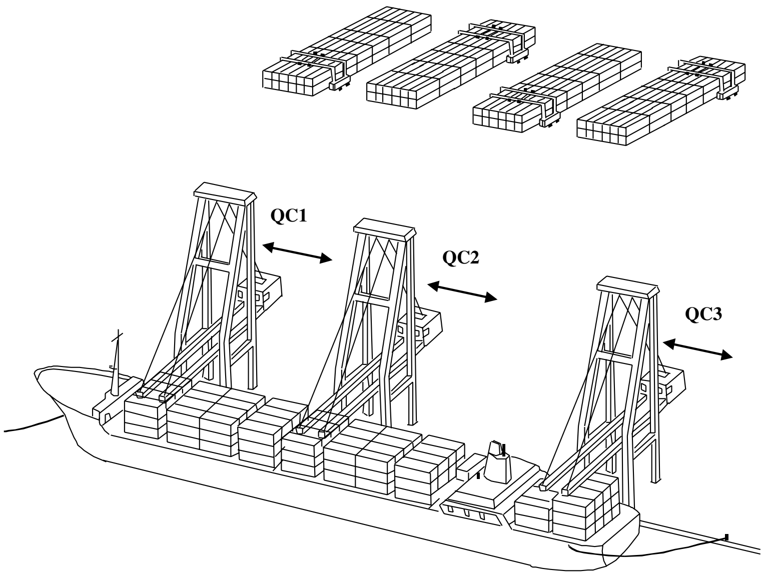

Fig. 1. A drawing of QCs working on a vessel.

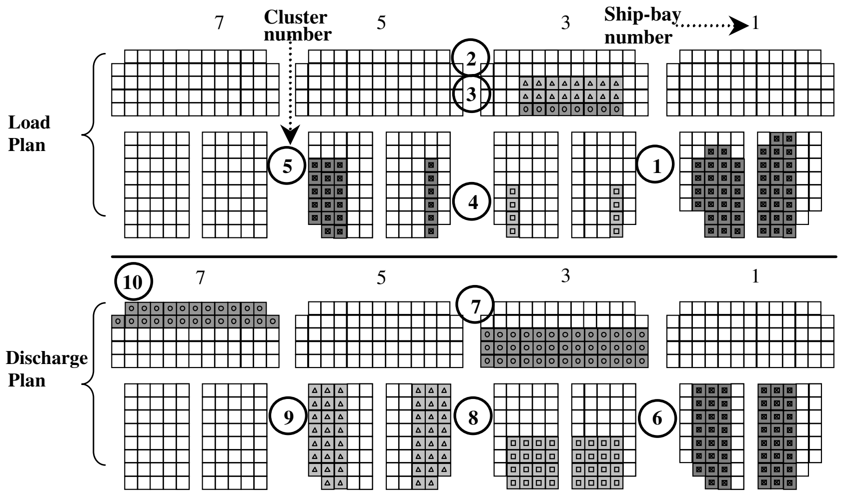

Fig. 2. A partial example of a stowage plan.

This paper defines a "task" as a discharging or loading operation for a cluster. This paper assumes that once a QC starts to load (or discharge) containers into (from) a cluster of slots, it continues to do so until all the slots in the cluster become filled (empty). Therefore, this paper considers handling work for a cluster to be a task.

Similar to the *m-parallel machine* scheduling problem (e.g., Guinet, 1993; Lee and Pinedo, 1997), the QC scheduling problem in this study addresses scheduling multiple QCs to perform discharging and loading operations in order to minimize the makespan of the operation. However, the QC scheduling problem has several unique characteristics that are different from those of a typical *m-parallel machine* problem. For

**Quay Crane Schedule**

| QC 1 (operation time: 09:00 ~ 12:00) | | | | | | | QC 2 (operation time: 09:00 ~ 12:00) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operation sequence | Cluster number | Location of task | Type of task | Number of containers | Start time | Finish time | Operation sequence | Cluster number | Location of task | Type of task | Number of containers | Start time | Finish time |
| **1** | 6 | 1 Hold* | D** | 47 | 09:00 | 09:47 | **1** | 7 | 3 Deck | D | 39 | 09:00 | 09:39 |
| **2** | 1 | 1 Hold | L** | 42 | 09:47 | 10:29 | **2** | 9 | 5 Hold | D | 46 | 09:41 | 10:26 |
| **3** | 8 | 3 Hold | D | 32 | 10:31 | 11:03 | **3** | 5 | 5 Hold | L | 23 | 10:26 | 10:49 |
| **4** | 4 | 3 Hold | L | 8 | 11:03 | 11:11 | **4** | 10 | 7 Deck | D | 24 | 10:51 | 11:14 |
| **5** | 3 | 3 Deck | L | 8 | 11:11 | 11:19 | | | | | | | |
| **6** | 2 | 3 Deck | L | 16 | 11:19 | 11:35 | | | | | | | |

* The hold of ship-bay 1.
** D (discharging), L (loading).

Fig. 3. An example of a QC schedule.

example, when discharging and loading operations must be performed at the same ship-bay, the discharging operation must precede the loading operation. When a discharging operation is performed in a ship-bay, tasks on a deck must be performed before tasks in the hold of the same ship-bay are performed. Also, the loading operation in a hold must precede the loading operation on the deck of the same ship-bay. Thus, there are precedence relationships among clusters, relationships that must be observed during a ship operation. Also, it should be noted that QCs travel on the same track. Thus, certain pairs of tasks cannot be performed simultaneously when the locations of the two clusters corresponding to the tasks are too close to each other, because two adjacent QCs must be apart from each other by at least one ship-bay so that they can simultaneously perform their tasks without interference. Also, if containers for any two tasks must be picked up at or delivered to the same location in a yard, the two tasks may not be performed simultaneously, because doing so will cause interference among yard cranes that transfer containers corresponding to the two tasks.

Fig. 3 illustrates a QC schedule that shows the number of containers in each cluster, the sequence of tasks to be performed, and the time schedule for performing the tasks.

## 3. A mathematical formulation

This section proposes a mathematical formulation for the QC scheduling problem. The constraints in the scheduling operations of QCs are shown below:

1. Each QC can operate after its earliest available time.
2. QCs are on the same track and thus cannot cross each other.
3. Some tasks must be performed before others.
4. There are some tasks that cannot be performed simultaneously.

The following notations are used for a mathematical formulation.

*Indices:*

| | |
|---|---|
| $i$, $j$ | Tasks to be performed. Tasks are ordered in an increasing order of their relative locations in the direction of increasing ship-bay numbers. |
| $k$ | QCs where $k = 1, \dots, K$. QCs are also ordered in an increasing order of their relative locations in the direction of increasing ship-bay numbers. |

*Problem data:*

| | |
|---|---|
| $p_i$ | The time required to perform task $i$. |
| $r_k$ | The earliest available time of QC $k$. |
| $l_i$ | The location of task $i$ (expressed by the ship-bay number). |
| $l_k^0$ | The starting position of QC $k$ expressed by a ship-bay number. |
| $l_k^T$ | The final position of QC $k$ expressed by a ship-bay number. This final position may be specified if a job for the next ship is already assigned to QC $k$. |
| $t_{ij}$ | The travel time of a QC from location ($l_i$) of task $i$ to location ($l_j$) of task $j$. $t_{0j}^k$ and $t_{jT}^k$ respectively represent the travel time from the initial position ($l_k^0$) of QC $k$ to location ($l_j$) of task $j$, and from location ($l_j$) of task $j$ to the final destination ($l_k^T$) of QC $k$. |
| $M$ | A sufficiently large constant. |
| $\alpha_1$ | The weight for the makespan (the maximum completion time). |
| $\alpha_2$ | The weight for the total completion time. |

*Sets of indices:*

| | |
|---|---|
| $\Omega$ | The set of all tasks. |
| $\Psi$ | The set of pairs of tasks that cannot be performed simultaneously. When tasks $i$ and $j$ cannot be performed simultaneously, $(i,j) \in \Psi$. |
| $\Phi$ | The set of ordered pairs of tasks between which there is a precedence relationship. When task $i$ must precede task $j$, $(i,j) \in \Phi$. |

*Decision variables:*

| | |
|---|---|
| $X_{ij}^k$ | 1, if QC $k$ performs task $j$ immediately after performing task $i$; 0, otherwise. Tasks 0 and $T$ will be considered to be the initial and final states of each QC, respectively. Thus, when task $j$ is the first task of QC $k$, $X_{0j}^k = 1$. Also, when task $j$ is the last task of QC $k$, $X_{jT}^k = 1$. |
| $Y_k$ | The completion time of QC $k$. |
| $D_i$ | The completion time of task $i$. |
| $Z_{ij}$ | 1, if task $j$ starts later than the completion time of task $i$; 0, otherwise. |
| $W$ | Time at which all tasks are completed. |

The QC scheduling problem can be formulated as follows:

$$\text{Minimize} \quad \alpha_1 W + \alpha_2 \sum_{k=1}^{K} Y_k, \tag{1}$$

subject to

$$Y_k \leqslant W \quad \forall k = 1, \ldots, K, \tag{2}$$

$$\sum_{j \in \Omega} X_{0j}^k = 1 \quad \forall k = 1, \ldots, K, \tag{3}$$

$$\sum_{i \in \Omega} X_{iT}^k = 1 \quad \forall k = 1, \ldots, K, \tag{4}$$

$$\sum_{k} \sum_{i \in \Omega} X_{ij}^k = 1 \quad \forall j \in \Omega, \tag{5}$$

$$\sum_{j} X_{ij}^k - \sum_{j} X_{ji}^k = 0 \quad \forall i \in \Omega,\ \forall k = 1, \ldots, K, \tag{6}$$

$$D_i + t_{ij} + p_j - D_j \leqslant M(1 - X_{ij}^k) \quad \forall i,j \in \Omega,\ \forall k = 1, \ldots, K, \tag{7}$$

$$D_i + p_j \leqslant D_j \quad \forall (i,j) \in \Phi, \tag{8}$$

$$D_i - D_j + p_j \leqslant M(1 - Z_{ij}) \quad \forall i,j \in \Omega, \tag{9}$$

$$Z_{ij} + Z_{ji} = 1 \quad \forall (i,j) \in \Psi, \tag{10}$$

$$\sum_{v=1}^{k} \sum_{u \in \Omega} X_{uj}^{v} - \sum_{v=1}^{k} \sum_{u \in \Omega} X_{ui}^{v} \leqslant M(Z_{ij} + Z_{ji}) \quad \forall i,j \in \Omega,\ l_i < l_j,\ \forall k = 1,\ldots,K, \tag{11}$$

$$D_j + t_{jT}^{k} - Y_k \leqslant M(1 - X_{jT}^{k}) \quad \forall j \in \Omega,\ \forall k = 1,\ldots,K, \tag{12}$$

$$r_k - D_j + t_{0j}^{k} + p_j \leqslant M(1 - X_{0j}^{k}) \quad \forall j \in \Omega,\ \forall k = 1,\ldots,K, \tag{13}$$

$$X_{ij}^{k}, Z_{ij} = 0 \text{ or } 1 \quad \forall i,j \in \Omega,\ \forall k = 1,\ldots,K, \tag{14}$$

$$Y_k, D_i \geqslant 0 \quad \forall i \in \Omega,\ \forall k = 1,\ldots,K. \tag{15}$$

In the objective function (1), it is assumed that $\alpha_1 \gg \alpha_2$, because the minimization of the makespan is considered to be more important than the minimization of the total completion time. This is a valid assumption because a ship can depart only after every QC assigned to it completes all the assigned tasks, and the earliest departure is the primary objective of QC scheduling. However, among possible solutions having the minimum makespan, the schedule with the shortest total completion time must be selected so that more QCs may become available to other ships as soon as possible. Constraint (2) evaluates the makespan. Constraints (3) and (4) respectively select the first and last tasks for each QC. Constraint (5) ensures that every task must be completed by exactly one QC. Constraint (6) is a flow balance constraint, guaranteeing that tasks are performed in well-defined sequences. Constraint (7) simultaneously determines the completion time for each task and eliminates sub-tours. When required, constraint (8) denotes that task $i$ should be completed before task $j$. Constraint (9) defines $Z_{ij}$ such that $Z_{ij} = 1$ when the operation for task $j$ starts after the operation for task $i$ is completed; 0, otherwise. Constraint (10) guarantees that tasks $i$ and $j$ cannot be performed simultaneously when $(i,j) \in \Psi$. By constraint (11), interference among QCs can be avoided. Suppose that tasks $i$ and $j$ are performed simultaneously and $l_i < l_j$. This means that $Z_{ij} + Z_{ji} = 0$. Note that both QCs and tasks are ordered in an increasing order of their relative locations in the direction of increasing ship-bay number. Suppose that, for $k_1 < k_2$, QC $k_1$ performs tasks $j$ and QC $k_2$ performs task $i$. Then, interference between QCs $k_1$ and $k_2$ results. However, in such a case, $\sum_{v=1}^{k_1} \sum_{u \in \Omega} X_{uj}^{v} - \sum_{v=1}^{k_1} \sum_{u \in \Omega} X_{ui}^{v} = 1$, which cannot be allowed because of constraint (11), and $Z_{ij} + Z_{ji} = 0$. The completion time of each QC is defined by constraint (12). Constraint (13) restricts the earliest starting time of operations by each QC.

The formulation of constraints (1)–(15) is a mixed-integer linear program. A computational experiment showed that the computational time is excessive for practical use. For example, a problem with two QCs and six tasks required 480 minutes to solve completely, using extended LINDO/PC (Release 6.1), which is a specialized software for integer and linear programs.

## 4. A branch and bound method

In the B & B method of this study, a solution is represented as a sequence of all tasks. Once a sequence of all tasks is given, a QC schedule can be constructed by assigning tasks to QCs in such a way that the first unassigned task in the sequence is assigned to the QC with the earliest completion time for already assigned tasks. This approach is called a "list scheduling." However, when evaluating the completion time of a QC, a delay of a QC resulting from interference with another QC must be considered. In list scheduling, no insertion of idle time is assumed. When compared to the formulation in Section 3, the solution space of this section is reduced by the above assumptions.

The B & B method essentially explores a solution tree starting from the root node at level 0 through nodes in the higher levels. The number of levels in the solution tree for the QC scheduling problem is the same as the number of all tasks. At each level, a task is selected during the enumeration process. Therefore, a node at level $i$ in the search tree corresponds to a partial solution that consists of the 1st through the $i$th tasks to be performed.

The following notations were used to describe the solution procedure.

| | |
|---|---|
| $W_k$ | The set of tasks assigned to QC $k$. |
| $S_k$ | The sequence of tasks assigned to QC $k$. |
| $c_k$ | The completion time of QC $k$ at the current node in the search tree. |
| $l_k^c$ | The position of QC $k$ after the last task in $S_k$ of the current node is performed. |

At the root node of the search tree, for all $k$, $W_k = \phi$, $S_k = (\ )$, $c_k = 0$, and $l_k^c = l_k^0$.

At each node, a solution is represented as $\{S_k\}_{1 \leqslant k \leqslant K}$. The overall procedure is shown in Fig. 4. A detailed explanation for each step is provided below.

*Select the next branching node until the first feasible solution is found*

Until the first feasible solution is found, a depth-first search is performed. That is, the next node for branching is selected among the newly sprouted nodes. The node with the minimum lower bound among the newly sprouted nodes is selected as the next branching node. The depth-first search is continued until the first feasible solution is found. The objective value of the first feasible solution becomes the initial value of the upper bound of the optimal objective value.

*Select the next branching node after the first feasible solution is found*

Once the first feasible solution is found by the depth-first search, all the nodes with partial solutions will be candidates for the next branching. The selection criteria for the next branching node is still the minimum lower bound.

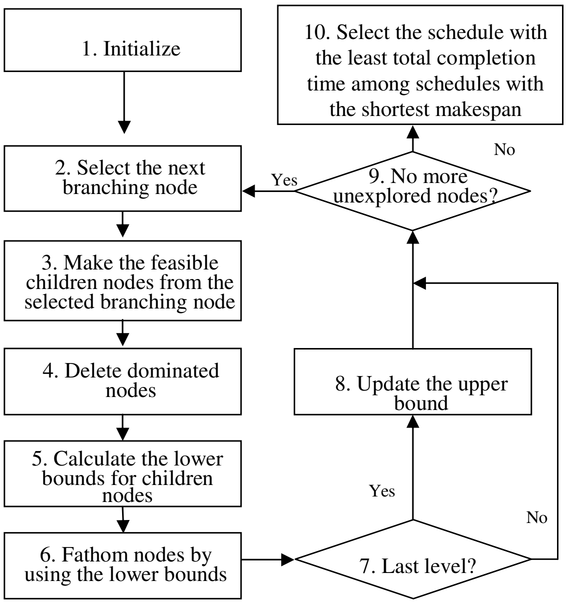
1. Initialize
2. Select the next branching node
3. Make the feasible children nodes from the selected branching node
4. Delete dominated nodes
5. Calculate the lower bounds for children nodes
6. Fathom nodes by using the lower bounds
7. Last level?
Yes
No
8. Update the upper bound
9. No more unexplored nodes?
Yes
No
10. Select the schedule with the least total completion time among schedules with the shortest makespan

Fig. 4. The overall procedure of the B & B method.

*Make feasible children nodes from the branching node*

First, among $S_j$, $j = 1, \dots, k$, the sequence, $S_k$, with the minimum $c_k$ is selected. Then, each remaining task will be appended to the sequence to make a child node. In the process, remaining tasks that violate the precedence constraints between tasks (constraint (8)) or the constraints regarding the interference between QCs (constraints (10) and (11)) are deleted.

*Delete dominated nodes*

Suppose that more than one node has the same elements of $\cup W_k$ and the value of $l_k^c$ for all $k$. Then, node $a$ is said to dominate node $b$ if node $a$ has a value of $c_k$ less than or equal to that of node $b$ for all $k$, and, for at least one $k$, node $a$ has a value of $c_k$ strictly lower than that of node $b$. All nodes dominated by another node are deleted.

*Calculate the lower bound of children nodes*

For each node, a lower bound is calculated and compared with the current upper bound of the optimal objective value. If the lower bound is greater than the current upper bound, the corresponding sub-tree connected to the node will no longer be explored.

When the locations of QCs and remaining tasks are given, an interval of $(l_k^c, l_{k+1}^c)$ is divided into several partitions by the positions of the remaining tasks, as shown in Fig. 5. Let the set of remaining tasks be $U(= \Omega - \cup_{k \in C} W_k)$ and the length of the $i$th partition be denoted as $u_{ki}$, which is expressed in units of travel time of QCs. Then, for a node in the search tree, a lower bound on the optimal objective value can be expressed as follows:

$$\text{Lower bound 1} = \max\{\max_k c_k, b_m\}, \tag{16}$$

where

$$b_m = \frac{1}{K}\left\{\sum_{k\in C} c_k + \sum_{i\in U} p_i + QC_{\text{travel}}\right\}, \tag{17}$$

$$QC_{\text{travel}} = t_l + t_b + t_r, \tag{18}$$

$$t_l = \max_{j\in U}\{l_1^c - l_j, 0\}, \tag{19}$$

$$t_r = \max_{j\in U}\{l_j - l_K^c, 0\}, \tag{20}$$

$$t_b = \sum_{k=1}^{K-1} u_k \tag{21}$$

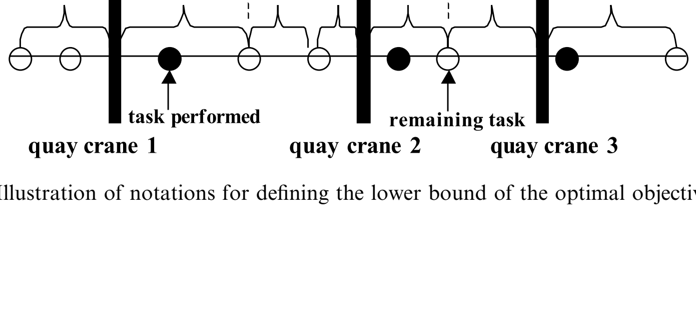

Fig. 5. Illustration of notations for defining the lower bound of the optimal objective value.

and

$$u_k = \sum_i u_{ki} - \max_i \{u_{ki}\}. \tag{22}$$

The following explains why expression (16) can be a lower bound on the optimal objective value. The value of $t_l$ is the lower bound on the travel time of a QC required to perform the left-most task, while $t_r$ is that required to perform the right-most task. Let an interval $(l_k^c, l_{k+1}^c)$ be divided into $m$ partitions according to the different positions of remaining tasks. Then, to complete the tasks between consecutive QCs, at least $(m-1)$ partitions must be covered by QC movements. Because $u_k$ is the minimum sum of lengths of $(m-1)$ partitions, $u_k$ becomes the lower bound on the total travel time of QCs required for performing tasks in the interval $(l_k^c, l_{k+1}^c)$. Thus, Eq. (21) is the lower bound on the travel time of QCs for completing all the remaining tasks between QC $k$ and $(k+1)$. Therefore, $(t_l + t_r + t_b)$ is the lower bound on the travel time to complete all the remaining tasks. Thus, $\{\sum_{k\in C} c_k + \sum_{i\in U} p_i + t_l + t_r + t_b\}$ is less than or equal to the minimum total completion time of QCs. Therefore, the optimal makespan must be greater than or equal to the total completion time, divided by the number of QCs. Thus, expression (16) is a lower bound on the optimal objective value.

*Lower bound* 2 is the same as *lower bound* 1 except that $QC_{\text{travel}}$ is evaluated as follows. While a lower bound on the minimum travel time of QCs is used as $QC_{\text{travel}}$ for *lower bound* 1, for *lower bound* 2, the exact minimum travel time of QCs is used as $QC_{\text{travel}}$. This problem of finding the exact minimum travel time is equivalent to the multiple traveling salesmen problem (MTSP). Because $QC_{\text{travel}}$ for lower bound 2 is the exact minimum travel time, *lower bound* 2 is tighter than *lower bound* 1. To find the optimal solution of MTSP, tasks must be optimally allocated to one of the QCs, and the optimal travel route for each QC for visiting the locations of all the allocated tasks must be determined.

By considering that all the tasks and QCs are located on a straight line, the optimal allocation of tasks to QCs has the form of solutions that can be obtained as follows. Partition each interval $(l_k^c, l_{k+1}^c)$ into left and right sub-intervals. Then, allocate tasks on the left sub-interval to QC $k$, and tasks on the right sub-interval to QC $(k+1)$. Thus, to find the optimal allocation, it is sufficient to enumerate all the possible partition points—which results in different allocations of tasks to QCs—between adjacent QCs. After all tasks are allocated to QCs, minimize the travel time as follows:

$$QC_{\text{travel}} = \sum_{k=1}^{K} t_k, \tag{23}$$

where

$$t_k = \max\{(l_k^c - l_{kl}), (l_{kr} - l_k^c)\} + 2\min\{(l_k^c - l_{kl}), (l_{kr} - l_k^c)\}. \tag{24}$$

In expression (24), $l_{kl}$ is the location of the left-most task among tasks in the left-hand side of the location of QC $k$, while $l_{kr}$ is the location of the right-most task among tasks in the right-hand side of the location of QC $k$. Suppose that $l_k^c - l_{kl} \leqslant l_{kr} - l_k^c$. Then, QC $k$ can minimize its travel time by moving to the location of the left-most task first and then moving to the location of the right-most task. In such a case, the travel time of QC $k$ will be $(l_{kr} - l_k^c) + 2(l_k^c - l_{kl})$. When $l_k^c - l_{kl} > l_{kr} - l_k^c$, the travel time of QC $k$ will be $(l_k^c - l_{kl}) + 2(l_{kr} - l_k^c)$. Note that $l_{kl}$ and $l_{kr}$ become $l_k^c$ when there is no allocated task in the left-hand side or the right-hand side of QC $k$, respectively.

*Stop the search process*

When there remains no more nodes with a partial solution, stop the process. Select the schedule with the smallest makespan. If there are multiple schedules with the smallest makespan, then select among them the schedule with the least total completion time.

The B & B method provides the optimal solution within a short computational time when the number of tasks is relatively small. However, the computational time increases rapidly when the problem size increases, which makes the B & B method difficult to use in practice. Therefore, GRASP, which is a heuristic procedure, is suggested, as discussed in Section 5.

## 5. A quay cranes scheduling procedure using greedy randomized adaptive search procedure

GRASP was developed in the late 1980s to solve combinatorial problems with high complexity as the set covering problem (Feo and Resende, 1995). Since then, it has been applied to a variety of combinatorial problems such as scheduling, graph-related problems, quadratic assignment problems, crew scheduling, machine and tool assignments, and location problems. The overall procedure of GRASP is summarized in Fig. 6.

GRASP consists of two phases: the solution construction phase and the solution improvement phase. In the solution construction phase, an element of a solution is added iteratively to an incomplete solution, and the value of the element is selected by using a random number and a greedy function. In the solution improvement phase, the constructed solution is locally improved until no more improvement is possible. Iterations are repeated a prespecified number of times. Then, the best solution is selected among all the obtained solutions. GRASP can be applied to the QC scheduling problem, as follows:

*Solution construction phase (Phase 1)*

*Step 1*: Among all QCs, the QC with the minimum $c_k$ is selected. A tie is broken randomly. Let QC $i$ be selected in this step.

*Step 2*: A set of feasible tasks for the next operation of QC $i$ is constructed. Let the set be denoted as $F$. Tasks that violate various constraints (the precedence constraint between tasks or interference among QCs) are excluded from $F$. Also, all the tasks $j \in F$ for which the following condition holds are also excluded from $F$:

$$1/|l_i^c - l_j| < r \max_{l \in F} \{1/|l_i^c - l_l|\}, \tag{25}$$

where $r$ is a constant between 0 and 1.

*Step 3*: Associate task $j \in F$ with the following probability:

```
procedure grasp()
    InputInstance();
    Repeat
        phase 1: ConstructGreedyRandomizedSolution(Solution);
        phase 2: LocalSearch(Solution);
        UpdateSolution(Solution,BestSolution);
    Until stopping criterion = true
    return(BestSolution);
end grasp
```

Fig. 6. A pseudo-code for GRASP.

$$(1/|l_i^c - l_j|)/\sum_{l\in F}(1/|l_i^c - l_l|). \tag{26}$$

*Step 4*: Randomly select a task $j \in F$ based on the probability distribution defined by (26).
*Step 5*: Steps 1–4 are repeated until all the tasks are assigned.

According to Eqs. (25) and (26), tasks that are located far from the location of a QC will be excluded from candidates for the next assignment to the QC (in case of (25)), or the tasks will have a low probability of being selected for the next assignment to the QC (in case of (26)).

*Solution improvement phase (Phase 2)*

*Step 0*: $i = 0$.
*Step 1*: $i = i + 1$. If $i > K$, stop. Otherwise, select QC $i$. Go to step 2.
*Step 2*: Select a pair of tasks that are assigned to QC $i$ and for which the highest improvement in $\max_k\{c_k\}$ can be obtained by performing a pairwise interchange (the 2-opt method). If there is more than one pair of tasks with the same improvement in $\max_k\{c_k\}$, select the task with the minimum total completion time. If the improvement is positive, perform the interchange. Repeat this step until no further improvement is possible. If no further improvement is possible, go to step 1.

Note that, even when only two tasks for the same QC is interchanged, the completion time of all the QCs must be evaluated again because the interference among QCs must be checked again. Even though more complicated improvement procedures (for example, 3-opt or interchange of tasks among different QCs) may be applied, the long computational time caused by the need to consider the interference among QCs make them feasible.

*The stopping criteria*
The solution construction procedure and the solution improvement procedure are performed repeatedly until the maximum number of iterations is reached or no better solution is found during a specified number of iterations (SNIs). When the stopping criteria is satisfied, the schedule with the minimum total completion time, among schedules with the shortest makespan, is selected as the final solution.
A numerical example of the QC scheduling procedure using GRASP is provided in the following. The input data from Table 1 was used.

Phase 1

(*Step 1*) QC 1 is selected ($c_1 = 0,\ c_2 = 0$).
(*Step 2*) $\Omega = \{1,2,3,4,5\}$, $F = \{1,2,3\}$
$1/|l_1^c - l_1| = 1/1 > 0.4 \times \max\{1/1, 1/2, 1/3\} = 0.4$
$1/|l_1^c - l_2| = 1/2 > 0.4 \times \max\{1/1, 1/2, 1/3\} = 0.4$

Table 1
Input data for the example

| Task ($i$) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_i$ | 8 | 10 | 20 | 5 | 30 |
| $l_i$ | 1 | 2 | 3 | 5 | 6 |
| $\Psi = \{(1,2),(2,3),(3,4),(4,5)\}$, $\Phi = \{\ \}$, $K = 2$, $l_1^c = 0$, $l_2^c = 4$, $r = 0.4$ | | | | | |

$1/|l_1^c - l_3| = 1/3 < 0.4 \times \max\{1/1, 1/2, 1/3\} = 0.4$
$F = \{1, 2\}$

(*Step 3*) According to Eq. (26), the probability of two tasks become as follows:
$((1/1)/(1/1 + 1/2), (1/2)/(1/1 + 1/2)) = (2/3, 1/3)$

(*Step 4*) Let the generated random number be 0.52. Job 1 is selected.
$c_1 = 1 + 8 = 9,\ c_2 = 0,\ l_1^c = 1,\ l_2^c = 4$

(*Step 1*) QC 2 is selected ($c_1 = 9,\ c_2 = 0$).

(*Step 2*) $\Omega = \{2, 3, 4, 5\}$, $F = \{3, 4, 5\}$
$1/|l_2^c - l_3| = 1/1 > 0.4 \times \max\{1/1, 1/1, 1/2\} = 0.4$
$1/|l_2^c - l_4| = 1/1 > 0.4 \times \max\{1/1, 1/1, 1/2\} = 0.4$
$1/|l_2^c - l_5| = 1/2 > 0.4 \times \max\{1/1, 1/1, 1/2\} = 0.4$
$F = \{3, 4, 5\}$

(*Step 3*) According to Eq. (26), the probability of tasks being selected become as follows:
$((1/1)/(1/1 + 1/1 + 1/2), (1/1)/(1/1 + 1/1 + 1/2), (1/2)/(1/1 + 1/1 + 1/2)) = (2/5, 2/5, 1/5)$

(*Step 4*) Let the generated random number be 0.71. Job 4 is selected.
$c_1 = 9,\ c_2 = 1 + 5 = 6,\ l_1^c = 1,\ l_2^c = 5$

(*Step 1*) QC 2 is selected ($c_1 = 9,\ c_2 = 6$).

...

...

(*Step 5*) All the tasks are assigned to QCs. The sequences of tasks for QCs 1 and 2 are $S_1 = (1, 3, 2)$ and $S_2 = (4, 5)$, respectively.

Phase 2

(*Step 1*) QC 1 is selected.

(*Step 2*) The neighboring sequence with the largest improvement is $S_1 = (1, 2, 3)$. No more improvement is possible. Go to step 1.

(*Step 1*) QC 2 is selected.

(*Step 2*) The neighboring sequence with the largest improvement is $S_2 = (5, 4)$. No more improvement is possible. Stop phase 2. The first iteration is ended.

## 6. A numerical experiment

Three different experiments were performed. The first experiment compared two lower bounds for the B & B method. Twenty-two problems (p13–p34 in Table 2) were solved by using lower bounds 1 and 2. Fig. 7 shows the ratio of the performance (the number of nodes enumerated and the computational time) of the B & B method using *lower bound* 2 to that using *lower bound* 1. Although the B & B method using *lower bound* 2 outperformed that using *lower bound* 1 based on the number of nodes enumerated, their computational times were similar. This is because *lower bound* 2 requires more time to evaluate.

The second experiment searched for the best set of parameters, (SNI, *r*) for GRASP, and the third experiment was to compared the performance of the B & B method with lower bound 1 with that of GRASP. For the second experiment, 12 problems (p1–p12), which have 2–3 QCs and 10–20 tasks, were generated. It was assumed that the number of ship-bays in the corresponding vessel was the same as the number of tasks. The initial locations of QCs were assumed to be equally spaced on the vessel. The operation time for a task was randomly generated from a uniform distribution of $U(3, 180)$. Also, attributes of tasks (hold task or deck task, loading or discharging) were chosen randomly. The precedence relationships among tasks in the same ship-bay—discharging tasks on deck, discharging tasks in a hold, loading tasks in a hold, and loading tasks on deck—were considered.

Table 2
The results of the computational experiment for problems p13–p49 ($r = 0.7$, SNI = 50)

| Test problems | | | B & B | | GRASP | | B & B/GRASP | |
|---|---|---|---|---|---|---|---|---|
| Problem no. | The number of QCs | Number of tasks | (1) Make-span | (2) Computation time (seconds) | (3) Make-span | (4) Computation time (seconds) | (3)/(1) | (4)/(2) |
| 13 | 2 | 10 | 453 | 20 | 453 | 28 | 1.00 | 1.40 |
| 14 | 2 | 10 | 546 | 17 | 546 | 17 | 1.00 | 1.00 |
| 15 | 2 | 10 | 513 | 13 | 516 | 11 | 1.01 | 0.85 |
| 16 | 2 | 10 | 321 | 64 | 321 | 15 | 1.00 | 0.23 |
| 17 | 2 | 10 | 456 | 28 | 456 | 23 | 1.00 | 0.82 |
| 18 | 2 | 10 | 375 | 35 | 375 | 31 | 1.00 | 0.89 |
| 19 | 2 | 10 | 552 | 44 | 552 | 22 | 1.00 | 0.50 |
| 20 | 2 | 10 | 480 | 17 | 480 | 22 | 1.00 | 1.29 |
| 21 | 2 | 10 | 465 | 1 | 465 | 16 | 1.00 | 16.00 |
| 22 | 2 | 10 | 720 | 22 | 720 | 26 | 1.00 | 1.18 |
| 23 | 2 | 15 | 576 | 372 | 591 | 65 | 1.03 | 0.17 |
| 24 | 2 | 15 | 669 | 368 | 675 | 94 | 1.01 | 0.26 |
| 25 | 2 | 15 | 738 | 2279 | 741 | 88 | 1.00 | 0.04 |
| 26 | 2 | 15 | 639 | 78 | 651 | 126 | 1.02 | 1.62 |
| 27 | 2 | 15 | 657 | 187 | 687 | 52 | 1.05 | 0.28 |
| 28 | 2 | 15 | 537 | 4010 | 549 | 83 | 1.02 | 0.02 |
| 29 | 2 | 15 | 807 | 651 | 819 | 94 | 1.01 | 0.14 |
| 30 | 2 | 15 | 891 | 12 | 906 | 97 | 1.02 | 8.08 |
| 31 | 2 | 15 | 570 | 2468 | 570 | 57 | 1.00 | 0.02 |
| 32 | 2 | 15 | 591 | 91 | 597 | 121 | 1.01 | 1.33 |
| 33 | 3 | 20 | 603 | 8973 | 666 | 110 | 1.10 | 0.01 |
| 34 | 3 | 20 | 717 | 8877 | 762 | 118 | 1.06 | 0.01 |
| 35 | 3 | 20 | 678–690[a] | 41014 | 699 | 127 | | 0.00 |
| 36 | 3 | 20 | 678–720 | 46355 | 708 | 238 | | 0.01 |
| 37 | 3 | 20 | 510–516 | 43961 | 540 | 279 | | 0.01 |
| 38 | 3 | 20 | 609–633 | 38749 | 660 | 160 | | 0.00 |
| 39 | 3 | 20 | 513–552 | 47125 | 579 | 234 | | 0.00 |
| 40 | 3 | 20 | 558–576 | 34473 | 597 | 148 | | 0.00 |
| 41 | 3 | 20 | 585–654 | 33893 | 642 | 269 | | 0.01 |
| 42 | 3 | 20 | 561–588 | 35264 | 666 | 214 | | 0.01 |
| 43 | 3 | 25 | 864–951 | 48938 | 942 | 291 | | 0.01 |
| 44 | 3 | 25 | 813–879 | 39649 | 858 | 619 | | 0.02 |
| 45 | 3 | 25 | 825–861 | 57568 | 873 | 399 | | 0.01 |
| 46 | 3 | 25 | 687–708 | 24506 | 735 | 667 | | 0.03 |
| 47 | 3 | 25 | 789–912 | 43687 | 807 | 437 | | 0.01 |
| 48 | 3 | 25 | 627–669 | 55966 | 669 | 307 | | 0.01 |
| 49 | 3 | 25 | 888–915 | 69771 | 972 | 457 | | 0.01 |

[a] An optimal solution could not be obtained because the number of nodes searched at a level exceeded 175,000, the preset limit in the program.

The GRASP algorithm was programmed by using Microsoft C 6.0 language on a Pentium II-466 MHz PC with 64 MB RAM. The algorithm was stopped when the number of iterations reached 500 or when no better solution was found during SNI iterations. In the experiment, the SNI was set to 10, 25, 50, 100, or 200. The value of $r$ in Eq. (25) was set to 0.1, 0.3, 0.5, 0.7, or 0.9. The experiment was conducted under 25 different combinations of parameters from five values of SNI and five values of $r$.

Among 12 test problems, only nine (p1–p9) could be solved by the B & B method. The nine problems were solved under 25 different experimental conditions. Fig. 8 shows that there was little change in the makespans when $r$ was less than 0.5. The makespan was lowest when $r$ was between 0.5 and 0.7. However,

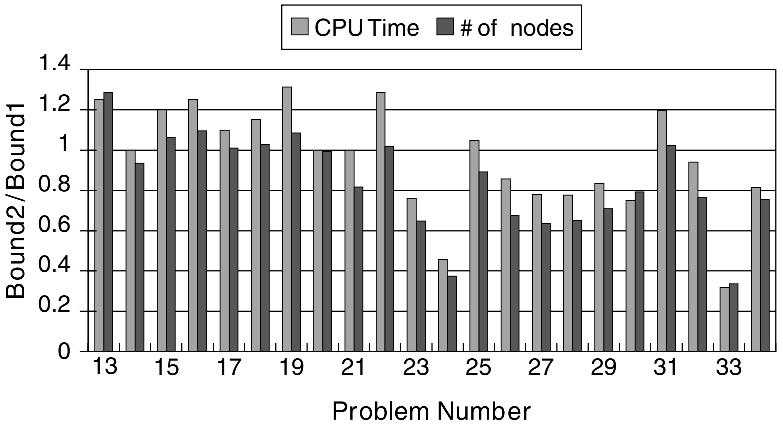

Fig. 7. A comparison of the number of enumerated nodes and the computational time for two lower bounds.

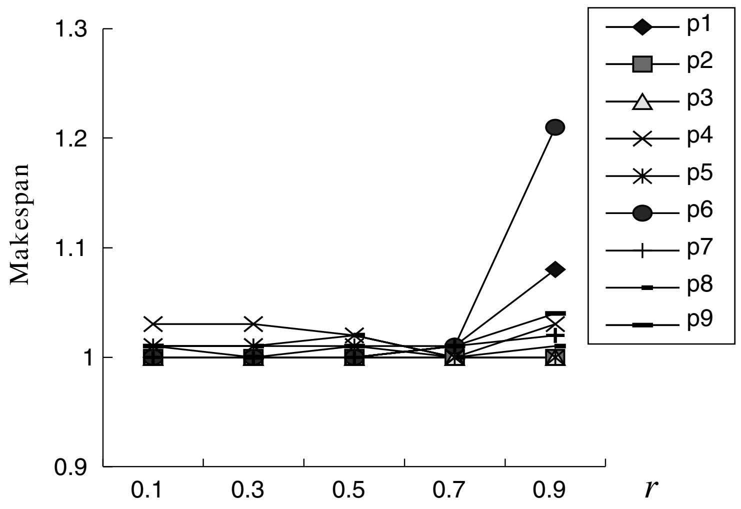

Fig. 8. Normalized makespans for various values of $r$ (SNI = 100).

the makespan increased dramatically when the value of $r$ equaled 0.9. Generally, a better solution was found with a larger value of SNI. We found that, for a low value of $r$, the improved amount in the objective value during the improvement phase increased as the SNI increased. In comparison, when the value of $r$ was high, the amount of improvement in the objective value was less sensitive to the change in the SNI. The algorithm became similar to a random search when the value of $r$ was very small. Through a preliminary experiment for the QC scheduling problem, it was concluded that the best value of $r$ is 0.7 (Fig. 8). Fig. 9 shows that the solution was improved as the value of the SNI increased. However, even at SNI = 10, the objective value resulting from GRASP did not exceed the optimal objective value by more than 10%. Also, Fig. 9 shows that the objective values found by GRASP became less than 103% of the optimal objective values after the SNI exceeded 100. Fig. 10 shows that the computational time was proportional to the value of the SNI.

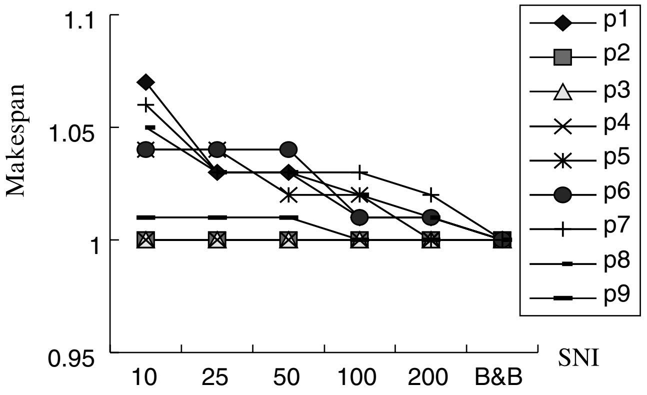

Fig. 9. Makespan for various values of SNI ($r = 0.7$).

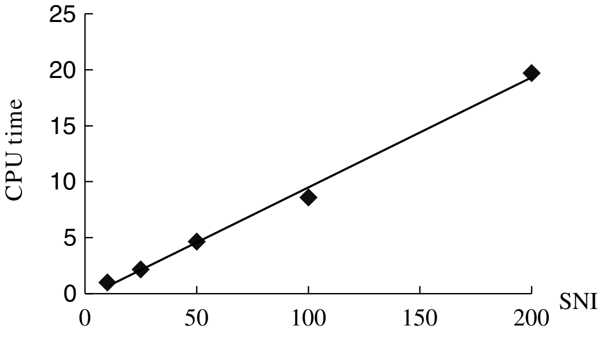

Fig. 10. The average CPU time for various values of SNI ($r = 0.7$, p1–p9). CPU time at SNI = 10 is normalized to be one.

The third experiment set SNI = 50 and $r = 0.7$. Ninety problems (p13–p102) were randomly generated as described previously. The number of QCs and tasks were assumed to be 2–6 and 10–50, respectively.

Table 2 compares makespans and computational times of p13–p49 between the B & B method and GRASP. For p13–p34, the B & B method obtained the optimal solution. However, for p35–p49, the B & B method could not obtain the optimal solution within a prespecified time. Thus, the search procedure of the B & B method was stopped when the number of nodes at a search level exceeded 175,000. For p35–p49 (the maximum lower bound, the minimum upper bound) obtained thus far were listed in the fourth column, and the computational times consumed before the stop of the search were listed in the fifth column. When the problem size is small, i.e., for p13–p22, GRASP also obtained optimal solutions. For problems 23–34, the objective values from GRASP were higher than those from the B & B method by less than 10%. However, the computational times of GRASP became less than 3% of those of the B & B method when the number of QCs and the number of tasks exceeded 3 and 20, respectively.

Based on the results of 90 problems (p13–p102), Fig. 11 shows how the computational time of GRASP changed as the number of tasks increased. It can be observed that the computational time increased rapidly as the number of tasks increased. Fig. 12 shows how the computational time changed as the number of QCs increased. It is interesting to observe that the computational time decreased as the number of QCs increased. Considering that, in practice, more QCs are usually used for larger numbers of tasks, the effect of the number of tasks on the computational time is offset by the effect of the number of QCs.

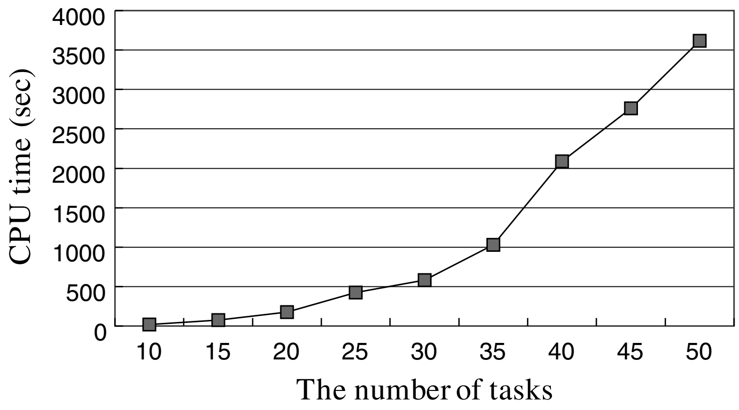

Fig. 11. The average CPU time of GRASP (SNI = 50, $r = 0.7$, p13–p102).

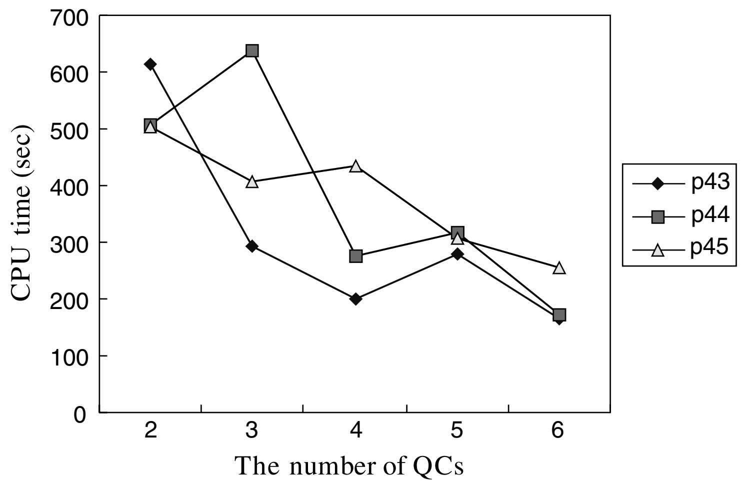

Fig. 12. The CPU time of GRASP for various numbers of QCs (SNI = 50, $r = 0.7$).

In addition, considering the fact that, in real QC scheduling problems, the number of QCs ranges from 2 to 6 and the number of tasks ranges from 10 to 50, GRASP can be considered as an appropriate methodology for solving practical QC scheduling problems.

## 7. Conclusion

This paper described a mathematical formulation for the QC scheduling problem, which is an important problem in the operation of port container terminals. To solve the problem, a B & B method and a lower bound of the optimal objective value were proposed. In addition, this study proposed the GRASP algorithm, a heuristic search algorithm with a function for escaping from a local minimum, to find near-optimal solutions to the QC scheduling problem. This study also performed a numerical experiment to evaluate the performance of the suggested algorithms. The final objective values derived by GRASP did not exceed those derived by the B & B method by more than 10% when the parameters of GRASP have values within specified ranges. Also, on the average, GRASP reduced the computational times to 3% of those of the B & B method when the number of QCs and the number of tasks exceeded 3 and 20, respectively. From a practical perspective, the computation time of GRASP was satisfactory.

It was noted that the ship planning process usually consists of several hierarchical levels of decision making such as berth planning, QC scheduling, and discharge and load sequencing. The decisions made at a higher level become constraints for decision making at a lower level. However, there are opportunities to improve the efficiency of plans by integrating the decision-making process at different levels. The integration of QC scheduling with berth planning or discharge and load sequencing may be possible with the help of new information technology, which is a promising topic for future study.

## Acknowledgements

The authors would like to thank an anonymous referee for his/her careful review and helpful suggestions. This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Center for Intelligent & Integrated Port Management Systems (CIIPMS) at Dong-A University.

## References

Brown, G.G., Lawphongpanich, S., Thurman, K.P., 1995. Optimizing ship berthing. Naval Research Logistics 41, 1–15.

Cheung, R.K., Li, C.-L., Lin, W., 2002. Interblock crane deployment in container terminals. Transportation Science 36 (1), 79–93.

Cho, D.W., 1982. Development of a methodology for containership load planning. Unpublished Ph.D. Dissertation. Oregon State University.

Cojeen, H.P., Dyke, P.V., 1976. The automatic planning and sequencing of containers for containership loading and unloading. In: Pitkin, Roche, Williams, (Eds.), Ship Operation Automation. North-Holland Publishing Co.

Daganzo, C.F., 1989. The crane scheduling problem. Transportation Research Part B 23B (3), 159–175.

Feo, T.A., Resende, M.G.C., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133.

Gifford, L.A., 1981. Containership load planning heuristic for a transtainer-based container port. Unpublished M.Sc. Thesis. Oregon State University.

Guinet, A., 1993. Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. International Journal of Production Research 31 (7), 1579–1594.

Kim, K.H., Kim, K.Y., 1999. An optimal routing algorithm for a transfer crane in port container terminals. Transportation Science 33 (1), 17–33.

Lee, Y.H., Pinedo, M., 1997. Scheduling jobs on parallel machines with sequence-dependent setup times. European Journal of Operational Research 100 (3), 464–474.

Lim, A., 1998. The berth planning problem. Operations Research Letters 22, 105–110.

Narasimhan, A., Palekar, U.S., 2002. Analysis and algorithms for the transtainer routing problem in container port operations. Transportation Science 36 (1), 63–78.

Peterkofsky, R.I., Daganzo, C.F., 1990. A branch and bound solution method for the crane scheduling problem. Transportation Research Part B 24 (3), 159–172.

Zhang, C., Wan, Y.-W., Liu, J., Linn, R.J., 2002. Dynamic crane deployment in container storage yards. Transportation Research Part B 36, 537–555.