9

An Intelligent Marshalling Plan Using a New Reinforcement Learning System for Container Yard Terminals

Yoichi Hirashima
Osaka Institute of Technology
Japan

1. Introduction

In recent years, the number of shipping containers grows rapidly, and in many container yard terminals, increasing throughput of material handling operation becomes important issue as well as decreasing the turnaround times of vessels. Material handling operations for loading containers into a vessel is highly complex, and the complexity grows at an exponential rate according to the growth of the number of containers, the operation occupy a large part of the total run time of shipping at container terminals. A challenge of this chapter is focused on improving throughput of the material handling operations for loading container on a vessel by using reinforcement learning. Commonly, materials are packed into containers and each container in a vessel has its own position determined by the destination, weight, owner, and so on (Siberholz et al., 1991; Günther & Kim, 2005). Thus, containers have to be loaded into a ship in a certain desired order because they cannot be rearranged in the ship. Therefore, containers must be rearranged before loading if the initial layout is different from the desired layout. Containers carried into the terminal are stacked randomly in a certain area called bay and a set of bays are called yard. The rearrangement process conducted within a bay is called marshalling.

In the problem, the number of stacks in each bay is predetermined and the maximum number of containers in a stack is limited. Containers are moved by a transfer crane and the destination stack for the container in a bay is selected from the stacks being in the same bay. In this case, a long series of container movements is often required to achieve a desired layout, and results that are derived from similar initial layouts can be quite different. Problems of this type have been solved by using techniques of optimization, such as genetic algorithm (GA) and multi agent method (Koza, 1992; Minagawa & Kakazu, 1997). These methods can successfully yield some solutions for block stacking problems. However, they adopt the environmental model different from the marshalling process, and do not assure to obtain the desired layout of containers.

Another candidate for solving the problem is the reinforcement learning (Watkins & Dayan, 1992), which is known to be effective for learning under unknown environment that has the

Markov Property. The Q-learning, one of the realization algorithm for the reinforcement learning can be applied to generate marshalling plan, when all the estimates of evaluation-values for pairs of the layout and container movement are obtained. These values are called ``Q-value''. The optimal series of container movements can be obtained by selecting the movement that has the best evaluation for each layout. However, conventional Q-learning has to store evaluation-values for all the layout-movement pairs. Therefore, the conventional Q-learning has great difficulties for solving the marshalling problem, due to its huge number of learning iterations required to obtain admissible plan (Baum, 1999). Recently, a Q-learning method that can generate marshalling plan has been proposed (Motoyama et al., 2001). Although these methods were effective for several cases, the desired layout was not achievable for every trial so that the early-phase performances of learning process can be spoiled. This chapter introduces a new Q-learning method for marshalling plan, and some additional methods to improve learning performances. The learning process in the proposed method is consisted of two stages: 1. determination of rearrangement order, 2. selection of destination for removal containers. Each stage has a corresponding learning algorithm, and Q-values in one stage are referred from the learning algorithm in the other stage. Stages are repeated sequentially in accordance with container movements and Q-values are discounted according to the number of container movements, so that Q-values reflect the total number of container movements. Consequently, selecting the best Q-values leads the best series of container movements required to obtain a desired layout. Moreover, each rearranged container is placed into the desired position so that every trial can achieve one of desired layouts. In addition, in the proposed method, each container has several desired positions in the final layout, and the feature is considered in the learning algorithm. Thus, the early-phase performances of the learning process can be improved.

The remainder of the chapter is organized as follows. The marshalling process in container yard terminals is elaborated in section 2, following the problem description. In section 3, a learning algorithm of the proposed method is detailed, and a data storage structure for storing Q-values is explained in this section. Computer simulations are conducted for several cases and proposed method is compared to conventional ones in section 4. Finally, concluding remarks are given in section 5.

2. Problem description

Fig.1 shows an example of container yard terminal. The terminal consists of containers, yard areas, yard transfer cranes, auto-guided vehicles, and port crane. Containers are carried by trucks and each container is stacked in a corresponding area called bay and a set of bays constitutes a yard area. Each bay has n_{y} stacks that m_{y} containers can be laden, the number of containers in a bay is k, and the number of bays depends on the number of containers. Each container is recognized by an unique name c_i $(i = 1, \cdots, k)$. A position of each container is discriminated by using discrete position numbers, $\boldsymbol{1}, \cdots, \boldsymbol{n}_{\mathrm{y}}\boldsymbol{m}_{\mathrm{y}}$. Then, the position of the container c_i is described by x_i $(1 \le i \le k, 1 \le x_i \le n_{\mathrm{y}}m_{\mathrm{y}})$, and the state of a bay is determined by the vector, $x = [x_1, \cdots, x_k]$.

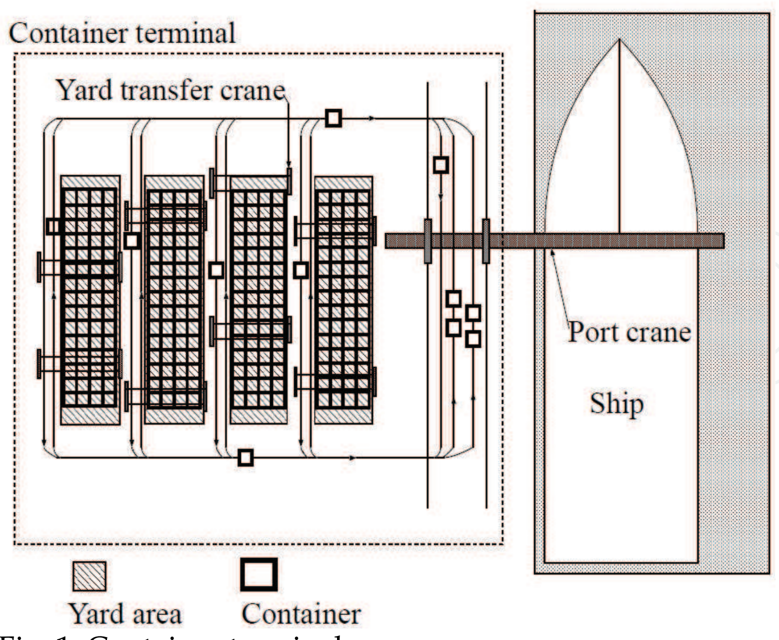

Fig. 1. Container terminal

2.1 Grouping

The desired layout in a bay is generated based on the loading order of containers that are moved from the bay to a ship. In this case, the container to be loaded into the ship can be anywhere in the bay if it is on top of a stack. This feature yields several desired layouts for the bay. Thus, in the addressed problem, when containers on different stacks are placed at the same height in a desired layout, it is assumed that the desired positions of such containers can be exchanged. Fig.2 shows an example of desired layouts, where $m_y = n_y = 3, k = 9$. In this example, containers are loaded in the ship in the descendent order. Then, containers c_7, c_8, c_9 are in the same group (Group1), and their positions are exchanged because the loading order can be kept unchanged after the exchange of positions. In the same way, c_5, c_5, c_6 are in the Group2, and c_1, c_2, c_3 are in the Group3 where positions of containers can be exchanged. Consequently several candidates for desired layout of the bay are generated from the original desired-layout.

In addition to the grouping explained above, a ``heap shaped group'' for n_y containers at the top of stacks in original the desired layout (group 1) is generated as follows:

1. n_y containers in group 1 can be placed at any stacks if their height is same as the original one.
2. Each of them can be stacked on other $n_y - 1$ containers when both of followings are satisfied:

   (a) They are placed at the top of each stack in the original desired-layout,
   (b) The container to be stacked is loaded into the ship before other containers being under the container.

Other groups are the same as ones in the original grouping, so that the grouping with heap contains all the desired layout in the original grouping.

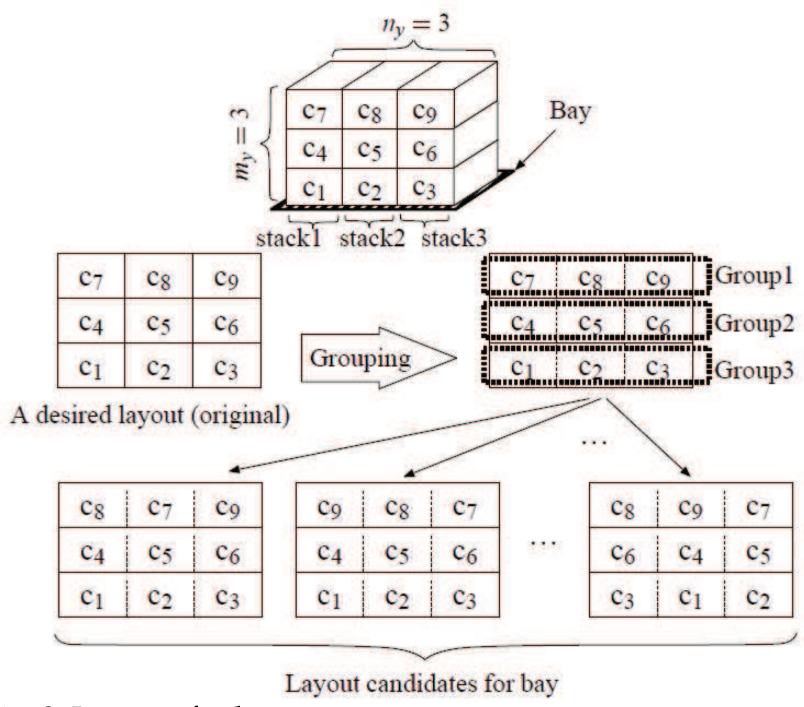

Fig. 2. Layouts for bay

2.2 Marshalling process

The marshaling process consists of 2 stages: ① selection of a container to be rearranged, and ② removal of the containers on the selected container in ①. After these stages, rearrangement of the selected container is conducted. In the stage ②, the removed container is placed on the destination stack selected from stacks being in the same bay. When a container is rearranged, n_{y} positions that are at the same height in a bay can be candidates for the destination. In addition, n_{y} containers can be placed for each candidate of the destination. Then, defining t as the time step, $\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})$ denotes the container to be rearranged at t in the stage ①. $\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})$ is selected from candidates $\mathrm{c}_{y_{i_1}}$ $(\boldsymbol{i}_1 = 1,\cdots,\boldsymbol{n}_{\mathrm{y}}{}^2)$ that are at the same height in a desired layout. A candidate of destination exists at a bottom position that has undesired container in each corresponding stack. The maximum number of such stacks is n_{y}, and they can have n_{y} containers as candidates, since the proposed method considers groups in the desired position. The number of candidates of $\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})$ is thus $\boldsymbol{n}_{\mathrm{y}} \times \boldsymbol{n}_{\mathrm{y}}$. In the stage ②, the container to be removed at t is $\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t})$ and is selected from two containers $\mathrm{c}_{y_{i_2}}$ $(i_2 = 1,2)$ on the top of stacks. In this stage, $\mathrm{c}_{\boldsymbol{y}_1}$ is on the $\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})$ and $\mathrm{c}_{\boldsymbol{y}_2}$ is on the destination of $c_{\mathrm{a}}(t)$. Then, in the stage ②, $c_{\mathrm{b}}(t)$ is removed to one of the other stacks in the same bay, and the destination stack *u*(*t*) at time *t* is selected from the candidates u_j ($j = 1, \cdots, n_{\mathrm{y}} - 2$). $c_{\mathrm{a}}(t)$ is rearranged to its desired position after all the $\mathrm{c}_{y_{i_2}}$ s are removed. Thus, a state transition of the bay is described as follows:

$$x_{t+1} = \begin{cases} f(x_t, c_{\mathrm{a}}(t)), & (\text{stage ①}) \\ f(x_t, c_{\mathrm{b}}(t), u(t)), & (\text{stage ②}) \end{cases} \tag{1}$$

where $f(\cdot)$ denotes that removal is processed and x_{t+1} is the state determined only by $c_{\mathrm{a}}(t)$, $c_{\mathrm{b}}(t)$ and *u*(*t*) at the previous state x_t. Therefore, the marshalling plan can be treated as the Markov Decision Process.

Additional assumptions are listed below:

a. The bay is 2-dimensional.
b. Each container has the same size.
c. The goal position of the target container must be located where all containers under the target container are placed at their own goal positions.
d. $k \le m_{\mathrm{y}} n_{\mathrm{y}} - 2m_{\mathrm{y}} + 1$

The maximum number of containers that must removed before rearrangement of $c_{\mathrm{a}}(t)$ is $2m_{\mathrm{y}} - 1$ because the height of each stack is limited to m_{y}. Thus, assumption d. assures the existence of space for removing all the $c_{\mathrm{b}}(t)$, and $c_{\mathrm{a}}(t)$ can be placed at the desired position from any state x_t.

Fig.3 shows 3 examples of marshalling process, where m_{y} =3, n_{y} =5, *k*=8. Positions of containers are discriminated by integers $1, \cdots, 15$. The first container to be loaded is c_8 and containers must be loaded by descendent order until c_1 is loaded. In the figure, a container marked with a □ denotes $c_{\mathrm{a}}(t)$, a container marked with a ○ is removed one, and an arrowed line links source and destination positions of removed container. Cases (a),(b) have the same order of rearrangement, $\mathrm{c}_2, \mathrm{c}_7, \mathrm{c}_6$, and the removal destinations are different. Whereas, case (c) has the different order of rearrangement, $\mathrm{c}_8, \mathrm{c}_2, \mathrm{c}_7$. When no groups are considered in desired arrangement, case (b) requires 5 steps to complete the marshalling process, and other cases require one more step. Thus, the total number of movements of container can be changed by the destination of the container to be removed as well as the rearrangement order of containers.

If groups are considered in desired arrangement, case (b) achieves a goal layout at step2, case (a) achieves at step3, case (c) achieves at step4. If extended groups are considered, cases (a),(b) achieve goal layouts at step2 and case (c) achieves at step4. Since extended goal layouts include the non-extended goal layouts, and since non-extended goal layouts include a non-grouping goal layout, equivalent or better marshalling plan can be generated by using the extended goal notion as compared to plans generated by other goal notions. The objective of the problem is to find the best series of movements which transfers every container from an initial position to the goal position. The goal state is generated from the shipping order that is predetermined according to destinations of containers. A series of movements that leads a initial state into the goal state is defined as an episode. The best episode is the series of movements having the smallest number of movements of containers to achieve the goal state.

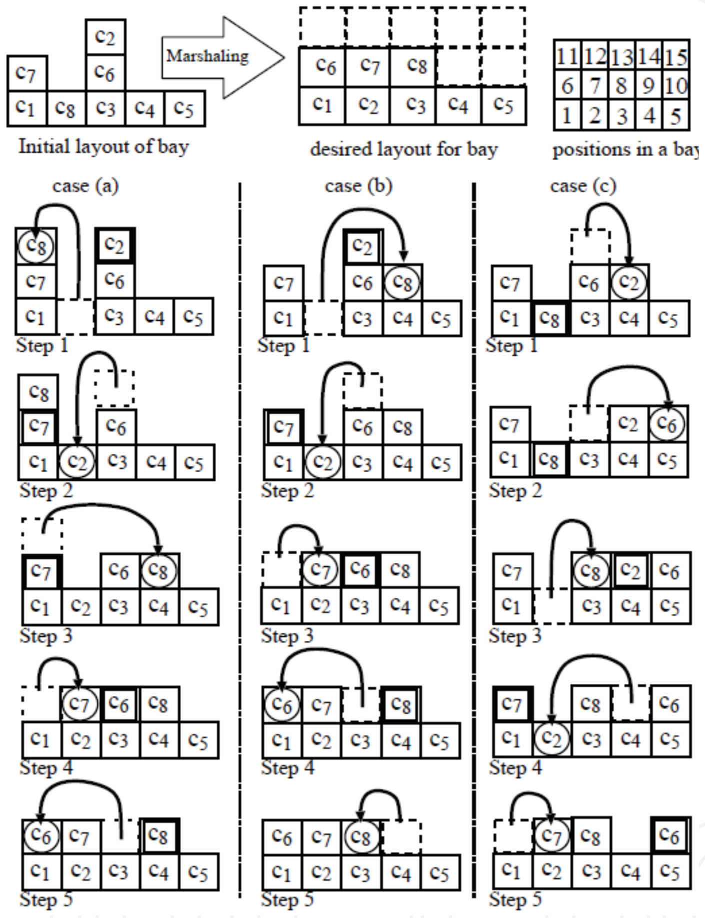

Fig. 3. Marshaling process

3. Reinforcement Learning for Marshalling Plan

3.1 Update rule of Q-values

In the selection of $\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})$, the container to be rearranged, an evaluation value is used for each candidate $\mathrm{c}_{y_{i_1}}$ $(\boldsymbol{i}_1 = 1 \cdots \boldsymbol{n}_{\mathrm{y}}^2)$. In the same way, evaluation values are used in the selection of the container to be removed $\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t})$ and its destination $\boldsymbol{u}_{\boldsymbol{j}}$ $(\boldsymbol{j} = 1 \cdots \boldsymbol{n}_{\mathrm{y}} - 2)$. Candidates of $\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t})$ is $\mathrm{c}_{y_{i_2}}$ $(\boldsymbol{i}_2 = 1,2)$. The evaluation value for the selection of $\mathrm{c}_{y_{i_1}}$, $\mathrm{c}_{y_{i_2}}$ and $\boldsymbol{u}_{\boldsymbol{j}}$ at the state $\boldsymbol{x}$ are called Q-values, and a set of Q-values is called Q-table. At the lth episode, the Q-value for selecting $\mathrm{c}_{y_{i_1}}$ is defined as $\boldsymbol{Q}_1(\boldsymbol{l},\boldsymbol{x},\mathrm{c}_{y_{i_1}})$, the Q-value for selecting $\mathrm{c}_{y_{i_2}}$ is defined as $\boldsymbol{Q}_2(\boldsymbol{l},\boldsymbol{x},\mathrm{c}_{y_{i_1}},\mathrm{c}_{y_{i_2}})$ and the Q-value for selecting $\boldsymbol{u}_j$ is defined as $\boldsymbol{Q}_3(\boldsymbol{l},\boldsymbol{x},\mathrm{c}_{y_{i_1}},\mathrm{c}_{y_{i_2}},\boldsymbol{u}_j)$. The initial value for $\boldsymbol{Q}_1,\boldsymbol{Q}_2,\boldsymbol{Q}_3$ is assumed to be 0.

In this method, a large amount of memory space is required to store all the Q-values referred in every episode. In order to reduce the required memory size, the length of episode that corresponding Q-values are stored should be limited, since long episode often includes ineffective movements of container. In the following, update rule of $\boldsymbol{Q}_3$ is described. When a series of n movements of container achieves the goal state $\boldsymbol{x}_n$ from an initial state $\boldsymbol{x}_0$, all the referred Q-values from $\boldsymbol{x}_0$ to $\boldsymbol{x}_n$ are updated. Then, defining L as the total counts of container-movements for the corresponding episode, $\boldsymbol{L}_{\min}$ as the smallest value of L found in the past episodes, and s as the parameter determining the threshold, $\boldsymbol{Q}_3$ is updated by the following equation when $L < \boldsymbol{L}_{\min} + s$ (s>0) is satisfied:

$$\boldsymbol{Q}_3(\boldsymbol{l}+1,\boldsymbol{x}_t,\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t}),\boldsymbol{u}_t) = (1-\alpha)\boldsymbol{Q}_3(\boldsymbol{l},\boldsymbol{x}_t,\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t}),\boldsymbol{u}_t) + \alpha[\mathrm{R} + \boldsymbol{V}_{t+1}], \tag{2}$$

$$\boldsymbol{V}_t = \begin{cases} \gamma \max\limits_{y_{i_1}} \boldsymbol{Q}_1(\boldsymbol{l},\boldsymbol{x},\mathrm{c}_{y_{i_1}}) & \text{(stage ①)}, \\ \gamma \max\limits_{y_{i_2}} \boldsymbol{Q}_2(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\mathrm{c}_{y_{i_2}}) & \text{(stage ②)}. \end{cases}$$

where γ denotes the discount factor and α is the learning rate. Reward R is given only when the desired layout has been achieved. $\boldsymbol{L}_{\min}$ is assumed to be infinity at the initial state, and updated by the following equation when $L < \boldsymbol{L}_{\min}$:
$L = \boldsymbol{L}_{\min}$.

In the selection of $\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t})$, the evaluation value $\boldsymbol{Q}_3(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t}),\boldsymbol{u}_j)$ can be referred for all the $\boldsymbol{u}_j$ ($\boldsymbol{j} = 1\cdots \boldsymbol{n}_{\mathrm{y}} - 2$), and the state $\boldsymbol{x}$ does not change. Thus, the maximum value of $\boldsymbol{Q}_3(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t}),\boldsymbol{u}_j)$ is copied to $\boldsymbol{Q}_2(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t}))$, that is,

$$\boldsymbol{Q}_2(\boldsymbol{l}+1,\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t})) = \max_{j} \boldsymbol{Q}_3(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\boldsymbol{c}_{\mathrm{b}}(\boldsymbol{t}),\boldsymbol{u}_j)\cdot \tag{3}$$

In the selection of $\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})$, the evaluation value $\boldsymbol{Q}_1(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}))$ is updated by the following equations:

$$\boldsymbol{Q}_1(\boldsymbol{l}+1,\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t})) = \begin{cases} \max\limits_{y_{i_1}} \boldsymbol{Q}_1(\boldsymbol{l},\boldsymbol{x},\mathrm{c}_{y_{i_1}}) + \mathrm{R} & \text{(stage ①)}, \\ \max\limits_{y_{i_2}} \boldsymbol{Q}_2(\boldsymbol{l},\boldsymbol{x},\boldsymbol{c}_{\mathrm{a}}(\boldsymbol{t}),\mathrm{c}_{y_{i_2}}) & \text{(stage ②)}. \end{cases} \tag{4}$$

In order to select actions, the "ε-greedy" method is used. In the "ε-greedy" method, $c_{\mathrm{a}}(t)$, $c_{\mathrm{b}}(t)$ and a movement that have the largest $Q_1(l,x,c_{\mathrm{a}}(t))$, $Q_2(l,x,c_{\mathrm{a}}(t),c_{\mathrm{b}}(t))$ and $Q_3(l,x,c_{\mathrm{a}}(t),c_{\mathrm{b}}(t),u_j)$ are selected with probability 1-ε ($0 < \varepsilon < 1$), and they are selected randomly with probability ε.

3.2 Learning algorithm

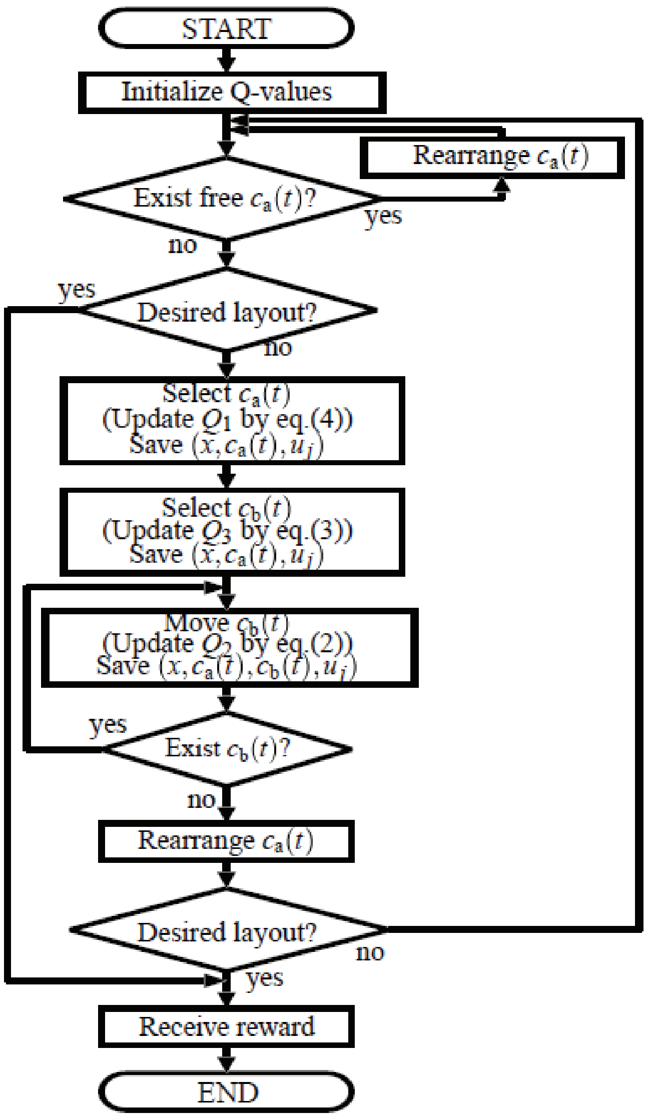

Fig. 4. Flowchart of the learning algorithm

By using the update rule, restricted movements and goal states explained above, the learning process is described as follows:

I. Count the number of containers being in the goal positions and store it as n
II. If $n = k$, go to X.
III. Select $c_{\mathrm{a}}(t)$ to be rearranged
IV. Store $(x, c_{\mathrm{a}}(t))$

V. Select $c_{\mathrm{b}}(t)$ to be removed

VI. Store $(x, c_{\mathrm{a}}(t), c_{\mathrm{b}}(t))$

VII. Select destination position u_j for $c_{\mathrm{b}}(t)$

VIII. Store $(x, c_{\mathrm{a}}(t), c_{\mathrm{b}}(t), u_j)$

IX. Remove $c_{\mathrm{b}}(t)$ and go to V. if another $c_{\mathrm{b}}(t)$ exists, otherwise go to I.

X. Update all the Q-values referred from the initial state to the goal state according to eqs. (2), (3)

A flow chart of the learning algorithm is depicted in Fig.4.

3.3 Data storage structure for storing Q-values

In the addressed problem, the state of a bay $x = [x_1, \cdots, x_k]$ is described by the positions of all the containers. In this case, the number of states of the bay increases by the exponential rate with increase of container counts. Also, evaluation values have to be stored for each state in order to compare candidates of, $c_{\mathrm{a}}(t)$, $c_{\mathrm{b}}(t)$, or u_j. In realistic situations the number of containers is often large, then required memory size to store information for all the state of the bay also becomes large.

Therefore, in the proposed method, binary trees for storing Q-values are constructed dynamically during the course of the learning, so that only Q-values corresponding states that are referred in learning process are stored (Hirashima et al., 1999). This feature can effectively reduce the required memory size for solving a marshalling problem and improve the solution. In the following, data storage structure of a lookup table for storing Q-values are explained.

A set of Q-values stored in a lookup table is called Q-table. In order to construct Q-table by using binary tree, the binary description of x_i ($i = 1 \cdots k$) is defined as $b_i = \beta_{i1} \cdots \beta_{iI}$ (β_{ij} =0,1; $j = 1, \cdots, \mathrm{I}$), where I is the order of binary description of x_i. Then, the binary description of x can be described by $B = b_1 \cdots b_k$ of order *k*I, and a binary tree of depth *k*I+1 is used to represent x. At each node of the binary tree, 0 is assigned to left descendant of the node and 1 is assigned to right descendant, and β_{ij} denotes the descendant at the node of depth I(*i*-1)+*j*. Each leaf of the tree stores state and corresponding Q-value. Given an input to the Q-table, the leaf corresponding to the input is specified by single search by using B. When the input corresponds to the value stored by the leaf, the Q-table outputs the Q-value stored by the leaf. Otherwise, the Q-table outputs 0. Fig.5 depicts a Q-table constructed by a binary tree in the case of $k = m_{\mathrm{y}} = n_{\mathrm{y}} = 2$, I=3. In the figure, inputs $x_{\varpi} = [1,3]$, $x_{\varepsilon} = [4,4]$ are given to the Q-table. Since b_i of x_{ϖ} is 001, descendants are specified by the order left, left and right from the root. Then, the leaf stores the same state as the input, and the Q-table outputs stored Q-value. While, b_i of x_{ε} is 100, descendants are specified by the order right and left from the root. Then, the state that leaf has is different from the input, and the Q-table outputs 0.

Initially, the tree has only root that has pointer to a leaf having data of state and Q-value. When the referred state has an updated Q-value, 2 consecutive memory units are newly allocated storing pointers to leafs storing data of state and Q-value. The Q-value and corresponding input are stored in another memory unit that is newly allocated for storing data according to β_{ij}. When the next updated Q-value appears, the input and the value pointed by the leaf are compared. When they have the same value, the stored Q-value is update. Otherwise 3 memory units are newly allocated in the memory space, one for data and others for pointers. The algorithm for Q-table construction is described below, and Fig.6 is the flowchart of the algorithm.

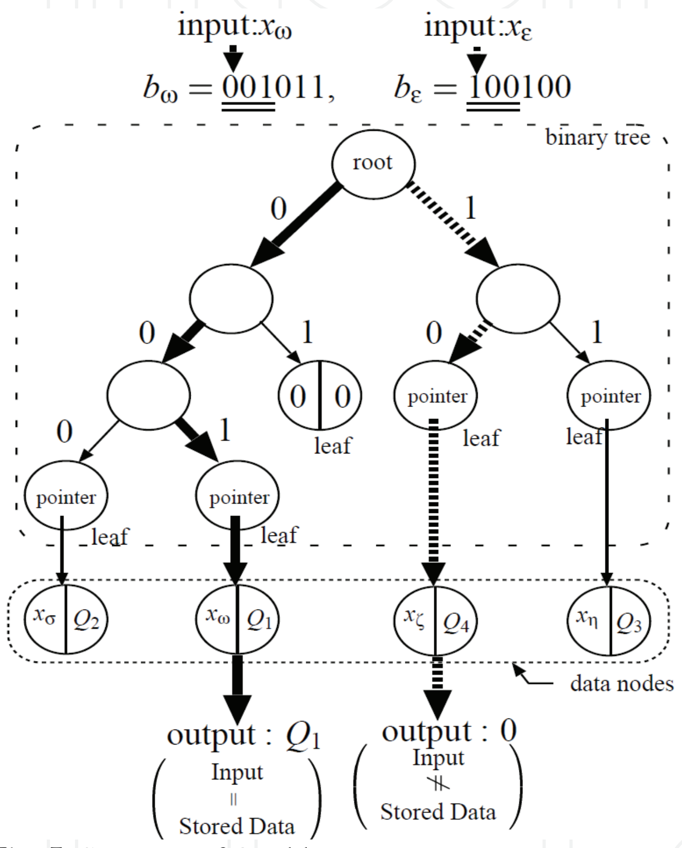

Fig. 5. Structure of Q-table

(1) Calculate $\boldsymbol{B}$ from $\boldsymbol{x}$ and initialize i=j=1

(2) If a memory unit corresponding to $\boldsymbol{B}$ is a leaf then go to 3, and if it is node then go to (4)

(3) update i,j by eq.(5)

$$\begin{cases} \boldsymbol{j} \leftarrow \boldsymbol{j}+1, \ \ \boldsymbol{i} \leftarrow \boldsymbol{i} \ \ (\boldsymbol{j} < \mathrm{I}) \\ \boldsymbol{j} \leftarrow 1, \boldsymbol{i} \leftarrow \boldsymbol{i}+1 \ \ (\boldsymbol{j} = \mathrm{I}) \end{cases} \tag{5}$$

and go to (2).

(4) Conduct eq.5 again, allocate 2 nodes for expanding a tree, and 1 leaf for storing state and Q-value. Then, copy data from original leaf into corresponding leaf, and store the pointers indicating a new leaf and nodes into original nodes.

(5) If β_{ij} has the same value as the state stored in the leaf, go to (4). Otherwise, store the new input and Q-value into the corresponding leaf.

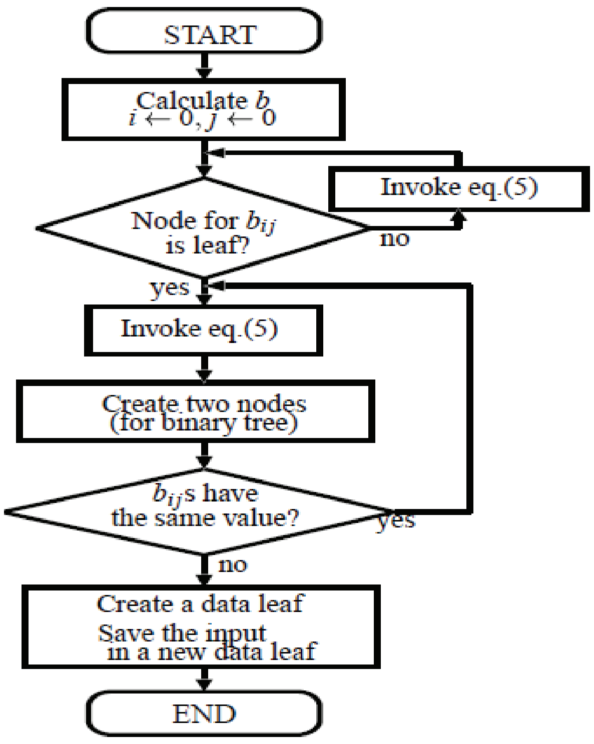
START
Calculate b
$i \leftarrow 0, j \leftarrow 0$
Invoke eq.(5)
Node for b_{ij} is leaf?
no
yes
Invoke eq.(5)
Create two nodes (for binary tree)
b_{ij}s have the same value?
yes
no
Create a data leaf
Save the input in a new data leaf
END

Fig. 6. Flowchart of the Q-table construction

4. Simulations

Computer simulations are conducted for 2 cases, and learning performances are compared for following 5 methods:

(A) proposed method considering grouping with heap,
(B) proposed method considering original grouping,
(C) a learning method using eqs. (2)-(4) as the update rule without grouping (Hirashima et al., 2005),
(D) method (E) considering original grouping.
(E) a learning method using, eqs. (2),(3) as the update rule, which has no selection of the desired position of $c_a(t)$ (Motoyama et al., 2001).

In methods (D),(E), although the stage $\textcircled{2}$ has the same process as the stage in the method (A), the container to be rearranged, $c_a(t)$, is simply selected from containers being on top of stacks. The learning process used in methods (D),(E) is as follows:

(i) The number of containers being on the desired positions is defined as k_B and count k_B
(ii) If $k_B = k$, go to (vi) else go to (iii),
(iii) Select $c_a(t)$ by using ε-greedy method,

(iv) Select a destination of $c_a(t)$ from the top of stacks by using ε-greedy method,
(v) Store the state and go to (i),
(vi) Update all the Q-values referred in the episode by eqs. (2),(3).

Since methods (D),(E) do not search explicitly the desired position for each container, each episode is not assured to achieve the desired layout in the early-phase of learning. The flowchart of the learning process in methods (D),(E) is described in Fig.7.

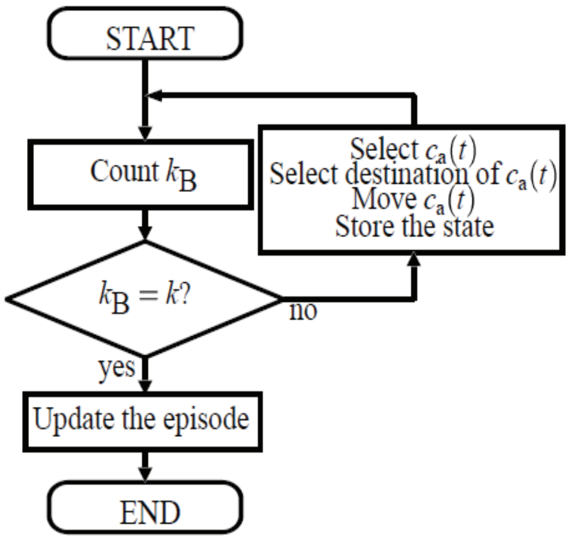

Fig. 7 Flowchart of learning process in methods (D),(E)

In methods (A)-(E), parameters in the yard are set as k=18, $m_y = n_y$ =6 that are typical values of marshalling environment in real container terminals. Containers are assumed to be loaded in a ship in descendant order from c_{18} to c_1. Fig.8 shows an original desired layout for the two cases, and Fig.9 shows corresponding initial layout for each case. Other parameters are put as α =0.8, γ =0.8, R=1.0, ε =0.8, s=15.

Results for case 2 are shown in Fig.10. In the figure, horizontal axis shows the number of trials, and vertical axis shows the minimum number of movements of containers found in the past trials. Each result is averaged over 20 independent simulations. In both cases, solutions that is obtained by methods (A),(B) and (C) is much better as compared to methods (D),(E) in the early-phase of learning, because methods (A),(B),(C) can achieve the desired layout in every trial, whereas methods (D),(E) cannot. Also, methods (A),(B) successfully reduces the number of trials in order to achieve the specific count of container-movements as compared to method (C), since methods (A),(B) considers grouping and finds desirable layouts that can easily diminish the number of movements of container in the early-phase learning. Moreover, at 10000th trail, the number of movements of containers in method (A) is smaller as compared to that in method (B) because, among the extended layouts, method (A) obtained better desired layouts for improving the marshalling process as compared to the layout generated by method (B). Desired layouts generated by methods (A),(B) are depicted in the Fig.11 for case 2.

	Case 1		Case 2	
Method	min. counts	ave. value	min. counts	ave. value
(A)	18	19.10	23	24.40
(B)	20	20.40	25	26.20
(C)	34	35.05	35	38.85
(D)	38	46.90	50	64.00
(E)	148	206.4	203	254.0

Table 1. The best solution of each method for cases 1, 2

The container-movement counts of the best solution and its averaged value for each method are described in Table1. Averaged values are calculated over 20 independent simulations. Among the methods, method (A) derives the best solution with the smallest container-movements. Therefore method (A) can improve the solution for marshalling as well as learning performance to solve the problem.

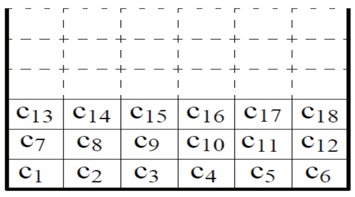

Fig. 8. A desired layout for cases 1,2

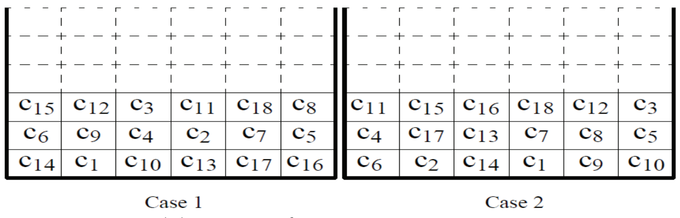

Fig. 9. Initial layouts for cases 1,2

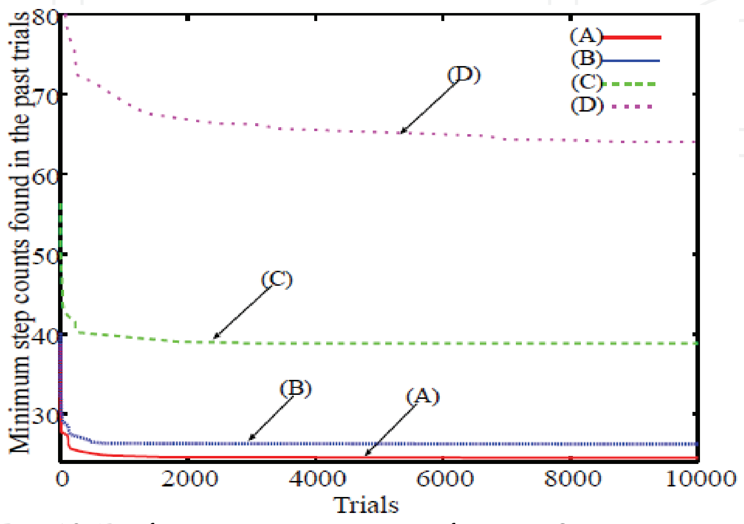

Fig. 10. Performance comparison for case 2

5. Conclusions

A new reinforcement learning system for marshalling plan at container terminals has been proposed. Each container has several desired positions that are in the same group, and the learning algorithm is designed to considering the feature.

In computer simulations, the proposed method could find solutions that had smaller number of movements of containers as compared to conventional methods. Moreover, since the proposed method achieves the desired layout in each trial as well as learns extended desirable layouts, the method can generate solutions with the smaller number of trials as compared to conventional methods.

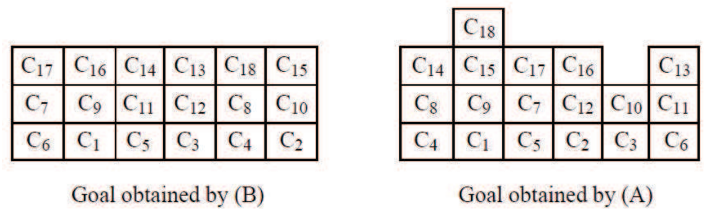

Fig. 11. Final layouts of the best solutions for case 2

6. References

Baum, E. B. (1999). Toward a model of intelligence as an economy of agents, *Machine Learning*, Vol. 35, 155–185.

Günther, H.-O. & Kim, K. H. (2005). *Container Terminals and Automated Transport Systems*, pp. 184–206, Springer.

Hirashima, Y., Iiguni, Y., Inoue, A., & Masuda, S. (1999). Q-learning algorithm using an adaptive-sized Q-table, *Proc. IEEE Conf. Decision and Control*, 1599–1604.

Hirashima, Y., takeda, K., Furuya, O., Inoue, A., & Deng, M. (2005). A new method for marshaling plan using a reinforcement learning considering desired layout of containers in terminals, *Preprint of 16th IFAC World Congress*, paperID We–E16–TO/2.

Koza, J. R. (1992). *Genetic Programming : On Programming Computers by means of Natural Selection and Genetics*, MIT Press.

Minagawa, M. and Kakazu, Y. (1997). An approach to the block stacking problem by multi agent cooperation, *Trans. Jpn. Soc. Mech. Eng. (in Japanese)*, C-63(608):231–240.

Motoyama, S., Hirashima, Y., Takeda, K., and Inoue, A. (2001). A marshalling plan for container terminals based on reinforce-ment learning, *Proc. of Inter. Sympo. on Advanced Control of Industrial Processes*, pages 631–636.

Siberholz, M. B., Golden, B. L., and Baker, K. (1991). Using simulation to study the impact of work rules on productivity at marine container terminals, *Computers Oper. Res.*, 18(5):433–452.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning, *Machine Learning*, 8:279–292.