A NEW METHOD FOR MARSHALING PLAN USING A REINFORCEMENT LEARNING CONSIDERING DESIRED LAYOUT OF CONTAINERS IN PORT TERMINALS

Yoichi HIRASHIMA* Osamu FURUYA* Kazuhiro TAKEDA**
Mingcong DENG* Akira INOUE*

**Department of Systems Engineering, Okayama University, 3-1-1 Tsushima-Naka,*
Okayama, 700-8530, Japan, hira suri.sys.okayama-u.ac.jp
***Hiroshima Research & Development Center, Mitsubishi Heavy Industries, LTD. ,*
Hiroshima, Japan

Abstract. In container yard terminals, containers brought by trucks in the random order. Containers have to be loaded into the ship in a certain order, since each container has its own shipping destination and it cannot be rearranged after loading. Therefore, containers have to be rearranged from the initial arrangement into the desired arrangement before shipping. In the problem, the number of container-arrangements increases by the exponential rate with increase of total count of containers, and the rearrangement process occupies large part of total run time of material handling operation at the terminal. Moreover, conventional methods require enormous time and cost to derive an admissible result for rearrangement process. In this paper, a Q-Learning algorithm considering the *desired position* of containers for a marshaling in the container yard terminal is proposed. In the proposed method, the learning process consists of two parts: rearrangement plan assuring explicit transfer of container to the *desired positin*, and, removal plan for preparing the rearrange operation. Using the proposed method, the learning performance can be improved as compared to the conventional method. In order to show effectiveness of the proposed method, computer simulations for several examples are conducted.

Keywords. Container marshaling, Block stacking problem, Q-learning, Reinforcement learning, Binary tree

1. INTRODUCTION

In recent years, operations for layout-rearrangement of container stacks occupy a large part of the total run time of shipping at container terminals. Since containers are moved by a transfer crane driven by human operator and thus, the container operation is important to reduce cost, run time, and environmental burden of material handling systems(Siberholz *et al.*, 1991). Commonly, materials are packed into containers and each container has its own shipping destination. Containers have to be loaded into a ship in a certain desired order because they cannot be rearranged in the ship. Thus, containers must be rearranged before loading if the initial layout is different from the desired layout. Containers carried in the terminal are stacked randomly in a certain area called bay and a set of bays are called yard area. When the number of containers for shipping is large, the rearrangement operation is complex and takes long time to achieve the desired layout of containers. Therefore the rearrangement process occupies a large part of the total run time of shipping. The rearrangement process conducted within a bay is called marshaling.

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 movements of containers is often required to achieve a desired layout, and results (the number of movements of container) that are derived from similar layouts can be quite different. Problems of this this type have been solved by using techniques of optimization, such as genetic algorithm (GA) and multi agent method(Koza, 1992; Minagawa and Kakazu, 1997). Although these methods yield some solutions, computational complexities are large, or, methods for improving the quality of solutions are not mentioned.

The Q-learning(Watkins and Dayan, 1992; Watkins, 1989) is known to be effective for learning under unknown environment. In the Q-learning for generating marshaling plan, all the estimates of evaluation-values for pairs of the layout and movement of containers are calculated. These values are called “Q-value” and Q-table is a look-up table that stores Q-values. The input of the Q-table is the plant state and the output is a Q-value corresponding to the input. A movement is selected with a certain probability that is calculated by using the magnitude of Q-values. Then, the Q-value corresponding to the selected movement is updated based on the result of the movement. The optimal pattern of container movements can be obtained by selecting the movement that has the largest Q-value at each state-movement pair, when Q-values reflect the number of container movements to achieve the desired layout. However, conventional Q-table has to store evaluation-values for all the state-movement pairs. Therefore, the conventional reinforcement learning method, Q-learning, has great difficulties for solving the marshaling problem, due to its huge number of learning iterations and states required to obtain admissible operation of containers(Baum, 1999). Recently, a Q-learning method that can generate marshaling plan has been proposed(Motoyama *et al.*, 2001). Although, these methods were effective several cases, the desired layout was not achievable for every trial so that the learning performances in early stages of learning can be degraded.

In this paper, a new reinforcement learning system to generate a marshaling plan is proposed. The learning process in the proposed method is consisted of two stages: (1) determination of rearrangement order, (2) selection of destination for removal containers. Learning algorithms in these stages are independent to each other and Q-values in one stage are referred from the other stage. That is, Q-values are discounted according to the number of container movement and Q-table for rearrangement is constructed by using Q-values for movements of container, so that Q-values reflect the total number of container movements required to obtain a desired layout. Moreover, in the end of stage (1), selected container is rearranged into the desired position so that every trial can achieve the desired layout. Thus, the learning performance in the early stages of learning can be improved. Also, in the addressed problem, the number of layout-movement pairs referred to achieve the desired layout is much smaller than the total number of layout-movement pairs. Thus, only the layout-movement pairs referred in learning processes are stored in Q-tables that are constructed dynamically(Hirashima *et al.*, 1999) by using the binary tree. Finally, effectiveness of the proposed method is shown by computer simulations for several cases.

2. PROBLEM DESCRIPTION

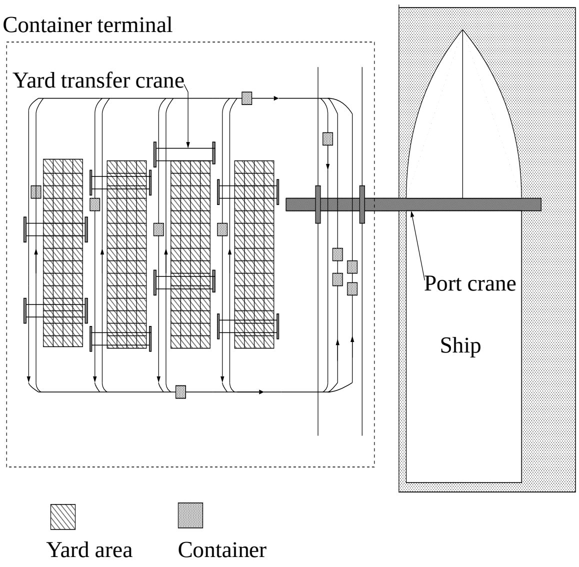

Fig. 1. Container terminal

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 $\mathrm{c}_i\,(i = 1, \cdots, k)$. A position of each container is discriminated by using discrete position numbers, $1, \cdots, n_{\mathrm{Y}} \cdot m_{\mathrm{Y}}$. Then, the position of the container c_i is described by $x_i\,(1 \leq i \leq k, 1 \leq x_i \leq m_{\mathrm{Y}} \cdot n_{\mathrm{Y}})$, and the state of a bay is determined by the vector, $\boldsymbol{x} = [x_1, \cdots, x_k]$. Defining t as the time step, the container to be rearranged at t in the stage (1) is $c_1(t)$ and is selected from candidates $\mathrm{c}_{y_{i_1}}\,(i_1 = 1, \cdots, n_{\mathrm{Y}})$, desired container to be placed at the bottom position that has undesired container in each stack. In the stage (2), the container to be rearranged at t is $c_2(t)$ and is selected from containers $\mathrm{c}_{y_{i_2}}\,(i_2 = 1, 2)$ being on the $c_1(t)$ and its desired-position. Then, in the stage (2), $c_2(t)$ 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 $u_j\,(j = 1, \cdots, n_{\mathrm{Y}} - 2)$. $c_1(t)$ is rearranged to its desired position after all the c_{y2_i} $(i = 1, 2)$ are removed. Thus, a state transition of the bay is described as follows:

$$\boldsymbol{x}_{t+1} = \begin{cases} f(\boldsymbol{x}_t, c_1(t)) & \text{(stage (1))} \\ f(\boldsymbol{x}_t, c_2(t), u(t)) & \text{(stage (2))} \end{cases} \tag{1}$$

where $f(\cdot)$ denotes that removal is processed and $\boldsymbol{x}_{t+1}$ is the state determined only by $c_1(t), c_2(t)$ and $u(t)$ at the previous state $\boldsymbol{x}_t$. Therefore, the marshaling plan can be treated as the Markov Decision Process.

Additional assumptions are listed below:

(1) The bay is 2-dimensional.
(2) Each container has the same size.
(3) The goal position of the target container must be located where all containers under the target container are placed at their own goal positions.
(4) $k \leq m_{\mathrm{Y}} n_{\mathrm{Y}} - 2m_{\mathrm{Y}} + 1$

The maximum number of containers that must removed before rearrangement of $c_1(t)$ is $2m_{\mathrm{Y}} - 1$ because the height of each stack is limited to m_{Y}. Thus, assumption (4) assures the existence of space for removing all the $c_2(t)$, and $c_1(t)$ can be placed at the desired position from any state $\boldsymbol{x}_t$.

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.

3. REINFORCEMENT LEARNING FOR MARSHALING PLAN

3.1 *Update rule of Q-values*

The evaluation value for the selection of c_{y_i} $(i = 1, \cdots, n_{\mathrm{Y}})$ and u_j $(j = 1, \cdots, n_{\mathrm{Y}}-2)$ at the state $\boldsymbol{x}$ is called Q-value, and a set of Q-values is called Q-table. At the *l*th episode, the Q-value for selecting $\mathrm{c}_{y_{i_1}}$ is defined as $Q_1(l, \boldsymbol{x}, u_j)$, the Q-value for selecting $\mathrm{c}_{y_{i_2}}$ is defined as $Q_2(l, \boldsymbol{x}, \mathrm{c}_{y_{i_1}}, \mathrm{c}_{y_{i_2}})$ and the Q-value for selecting u_j is defined as $Q_3(l, \boldsymbol{x}, \mathrm{c}_{y_{i_1}}, \mathrm{c}_{y_{i_2}}, u_j)$. The initial value for both Q_1, Q_2, 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 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, L_{min} as the smallest value of L found in the past episodes, and s as the parameter determining the threshold, Q_3 is updated when $L < L_{min} + s\,(s > 0)$ is satisfied by the following equation:

$$\begin{aligned} & Q_3(l, \boldsymbol{x}_t, c_1(t), c_2(t), u(t)) = \\ & \quad (1-\alpha)Q_3(l-1, \boldsymbol{x}_t, c_1(t), c_2(t), u(t)) \\ & \quad +\alpha[R + V_{t+1}] \\ & V_t = \begin{cases} \gamma \max\limits_{y_{i_1}} Q_1(l, \boldsymbol{x}_t, \mathrm{c}_{y_{i_1}}) & \text{(stage(1))} \\ \gamma \max\limits_{y_{i_2}} Q_2(l, \boldsymbol{x}_t, c_1(t), \mathrm{c}_{y_{i_2}}) & \text{(stage(2))} \end{cases} \end{aligned} \tag{2}$$

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

In the selection of $c_2(t)$, the evaluation value $Q_3(l, \boldsymbol{x}, c_1(t), c_2(t), u_j)$ can be referred for all the u_j $(j = 1 \cdots n_{\mathrm{Y}} - 2)$, and the state $\boldsymbol{x}$ does not change. Thus, the maximum value of $Q_3(l, \boldsymbol{x},\ c_1(t), c_2(t), u_j)$ is copied to $Q_1(l, \boldsymbol{x}, c(t))$, that is,

$$\begin{aligned} & Q_2(l, \boldsymbol{x}, c_1(t), c_2(t)) = \\ & \quad \max_j Q_3(l, \boldsymbol{x}, c_1(t), c_2(t), u_j). \end{aligned} \tag{3}$$

In the selection of $c_1(t)$, the evaluation value $Q_1(l, \boldsymbol{x}, c_1(t))$ is updated by the following equations:

$$\begin{aligned} & Q_1(l, \boldsymbol{x}_t, c_1(t)) = \\ & \begin{cases} \max\limits_{y_{i_1}} Q_1(l, \boldsymbol{x}_t, \mathrm{c}_{y_{i_1}}) + R & \text{(stage(1))} \\ \max\limits_{y_{i_2}} Q_2(l, \boldsymbol{x}_t, c_1(t), \mathrm{c}_{y_{i_2}}) & \text{(stage(2))} \end{cases} \end{aligned} \tag{4}$$

In the proposed method, the ”ϵ-greedy” is used to select actions. In the ”ϵ-greedy” method, $c_1(t), c_2(t)$ and a movement that have the largest $Q_1(l, \boldsymbol{x}, c_1(t))$, $Q_2(l,\ \boldsymbol{x}, c_1(t), c_2(t)$ and $Q_2(l, \boldsymbol{x}, c_1(t), c_2(t), u_j)$ are selected with probability $1 - \epsilon(0 < \epsilon < 1)$, and with probability ϵ, a container and a movement are selected randomly.

3.2 *Learning algorithm*

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

(1) Count the number of containers being in the goal positions and store it as n
(2) If $n = k$, go to (10)
(3) Select $c_1(t)$ to be rearranged
(4) Store $(\boldsymbol{x}, c_1(t))$
(5) Select $c_2(t)$ to be removed
(6) Store $(\boldsymbol{x}, c_1(t), c_2(t))$
(7) Select destination position u_j for $c_2(t)$
(8) Store $(\boldsymbol{x}, c_1(t), c_2(t), u_j)$
(9) Remove $c_2(t)$ and go to (5) if another $c_2(t)$ exists, otherwise go to (1)
(10) Update all the Q-values referred from the initial state to the goal state according to eqs. (2), (3)

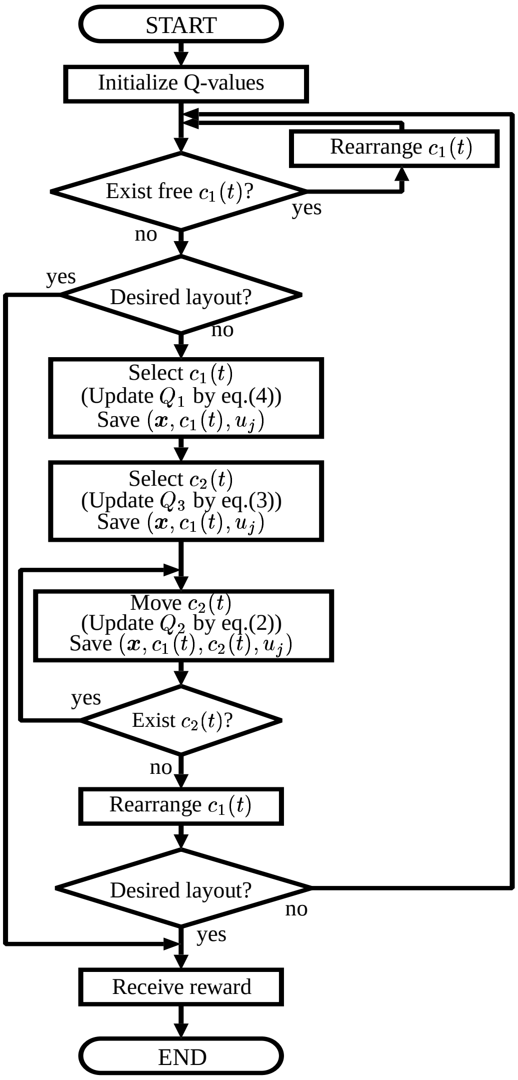

Fig. 2. Flowchart of the learning algorithm

A flow chart of the learning algorithm is depicted in Figure 2.

3.3 *Data structure of Q-table*

In the Q-learning method, a larger number of states result in the larger number of Q-values that should be updated. In this case, the number of learning iterations becomes larger. Moreover, the larger number of Q-values requires the lager memory size in order to store all the evaluation-values for pairs of the plant state and movement. In realistic problems the number of state-movement pairs is large, so that huge memory size is required in order to store Q-values for all the states. Therefore, in the proposed method, only Q-values corresponding states that are referred in learning process are stored. Binary trees for storing Q-values are constructed dynamically during the course of the learning.

The binary description of $x_i\,(i = 1, \cdots, k)$ is defined as $\boldsymbol{b}_i = b_{i1} \cdots b_{iI}\,(b_{ij} = 0, 1 \quad j = 1, \cdots, I)$, where I is the order of binary description of x_i. Then, the binary description of $\boldsymbol{x}$ can be described by $\boldsymbol{b} = \boldsymbol{b}_1 \cdots \boldsymbol{b}_k$ of order $(k+1)I$. That is, a binary tree of depth $(k+1)I$ can represent $\boldsymbol{x}$. At each node of the binary tree, 0 is assigned to left descendant of the node and 1 is assigned to right descendant, and b_{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 a search using $\boldsymbol{b}$. When the input corresponds the value stored by the leaf, Q-table outputs the Q-value stored by the leaf. Otherwise, Q-table outputs 0. Fig.3 depicts a Q-table constructed by binary tree in the case of $k = n_{\mathrm{y}} = m_{\mathrm{y}} = 2, l = 1, I = 3$. In the figure, inputs $\boldsymbol{x}_\omega = [1, 3], \boldsymbol{x}_\epsilon = [4, 4]$ are given to the Q-table. In the former case, left, left, right descendants are specified, the leaf stores the same state as the input, and outputs Q_1. While, in the latter case, right, left descendants are specified, the state that leaf has is different from the input, and thus 0 is output.

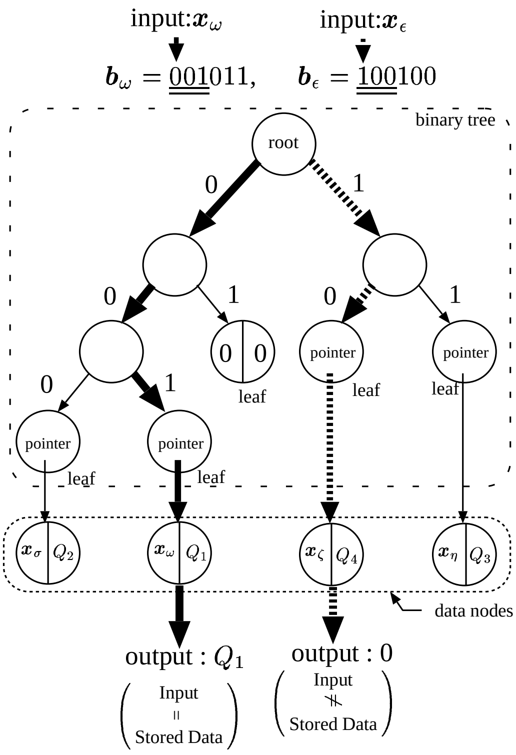
input:$\boldsymbol{x}_\omega$
input:$\boldsymbol{x}_\epsilon$
$\boldsymbol{b}_\omega = \underline{\underline{001}}011,$
$\boldsymbol{b}_\epsilon = \underline{\underline{100}}100$
binary tree
root
0
1
0
1
0
1
0 0
leaf
pointer
leaf
pointer
leaf
0
1
pointer
leaf
pointer
leaf
$\boldsymbol{x}_\sigma$ Q_2
$\boldsymbol{x}_\omega$ Q_1
$\boldsymbol{x}_\zeta$ Q_4
$\boldsymbol{x}_\eta$ Q_3
data nodes
output : Q_1
$\begin{pmatrix}\text{Input} \\ = \\ \text{Stored Data}\end{pmatrix}$
output : 0
$\begin{pmatrix}\text{Input} \\ \neq \\ \text{Stored Data}\end{pmatrix}$

Fig. 3. Structure of Q-table

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 for storing pointer to leafs storing data of state and Q-value are newly allocated. The Q-value and corresponding input are stored in another memory unit that is newly allocated for storing data according to b_{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.4 is the flowchart of the algorithm.

(1) Calculate $\boldsymbol{b}$ from $\boldsymbol{x}$ and initialize $i = j = 1$
(2) If a memory unit corresponding to b_{ij} is a leaf then go to (3), and if it is node then go to (4)
(3) update i, j by

$$\begin{cases} j \leftarrow j+1, \; i \leftarrow \quad i \quad (j < I), \\ j \leftarrow \quad 1, \quad i \leftarrow i+1 \; (j = I), \end{cases} \qquad (5)$$

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

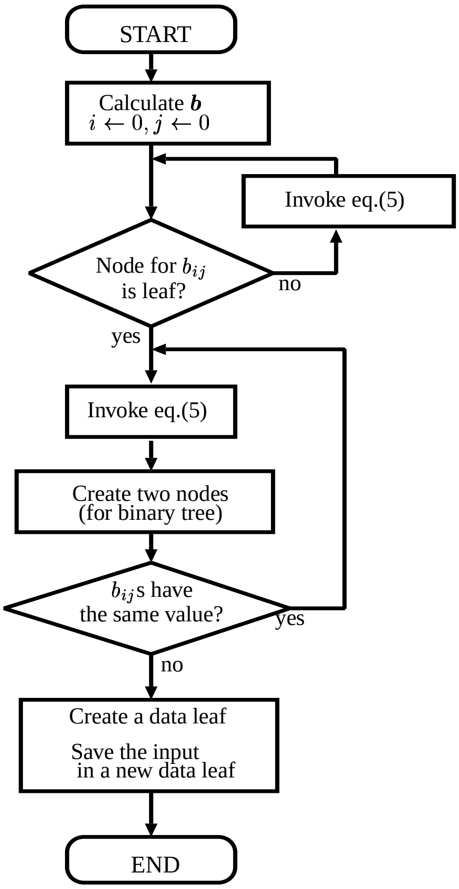
START
Calculate $\boldsymbol{b}$
$i \leftarrow 0, j \leftarrow 0$
Invoke eq.(5)
Node for b_{ij} is leaf?
no
yes
Invoke eq.(5)
Create two nodes (for binary tree)
b_{ij}s have the same value?
yes
no
Create a data leaf
Save the input in a new data leaf
END

Fig. 4. Flowchart of the Q-table construction

4. SIMULATIONS

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

(A) proposed method,
(B) a learning method using eqs. (2),(3) as the update rule, which has no selection of the desired position of $c_1(t)$(Motoyama *et al.*, 2001).

In the method (B), although the stage (2) has the same process as in the method (A), the container to be rearranged, $c_1(t)$, is simply selected from containers being on top of stacks. The learning process used in the method (B) is as follows:

(1) The number of containers being on the desired positions is defined as k_{B} and count k_{B}
(2) If $k_{\mathrm{B}} = k$, go to (6) else go to (3),
(3) Select $c_1(t)$ by using ϵ-greedy method,
(4) Select a destination of $c_1(t)$ from the top of stacks by using ϵ-greedy method,
(5) Store the state and go to (1),
(6) Update all the Q-values referred in the episode by eqs. (2),(3).

Since the method (B) does not search explicitly the desired position for each container, each epsode is not assured to achieve the desired layout in the early stages of learning. The flowchart of the learning process in the method (B) is described in Figure 5.

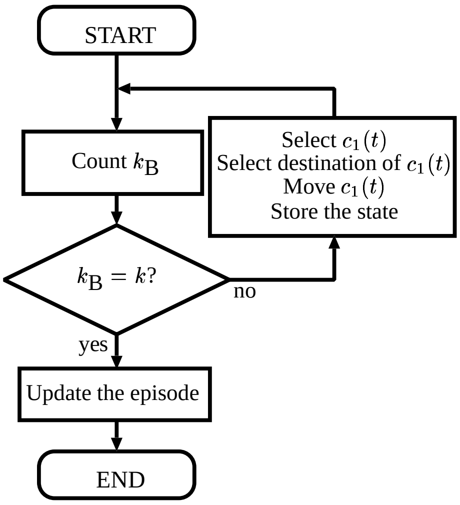

Fig. 5. Flowchart of the Q-table construction

In both methods, $k = 18, m_{\mathrm{Y}} = n_{\mathrm{Y}} = 6$ that is typical marshaling environment in real container terminals. Containers are assumed to be loaded in a ship in descendant order from c_{18} to c_1. Figure 6 shows the desired layout for the two cases, and figure 7 shows the initial layout for each case. Other parameters are put as $\alpha = 0.8, \gamma = 0.8, R = 1.0, \epsilon = 0.8, s = 15$.

Results are shown in Figs. 8,9. In the figures, 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 10 independent simulations. In both cases the method (A) obtain better solutions with smaller number of trials as compared to the method (B) in the early stages of learning, because the method (A) can achieve the desired layout in every trial, whereas the method (B) cannot. Moreover, at 10000th trail the number of movements of containers in the method (A) is coequal or better as compared to that in the method (B). Therefor the learning performance of the method

(A) has been improved. Although method (B) can occasionally generate comparative plan with method (A), the method (B) may generate the marshaling plan that cannot achieve the desired layout. This means that method (B) requires additional algorithm for checking achievability of the goal state.

c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}
c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
c_1	c_2	c_3	c_4	c_5	c_6

Fig. 6. Desired layout for cases 1,2

Initial layouts:

c_{15}	c_{12}	c_3	c_{11}	c_{18}	c_8
c_6	c_9	c_4	c_2	c_7	c_5
c_{14}	c_1	c_{10}	c_{13}	c_{17}	c_{16}

Case 1

c_{11}	c_{15}	c_{16}	c_{18}	c_{12}	c_3
c_4	c_{17}	c_{13}	c_7	c_8	c_5
c_6	c_2	c_{14}	c_1	c_9	c_{10}

Case 2

Fig. 7. Initial layouts for cases 1,2

The run time of a simulation was about 4 minutes by using the computer that have Pentium III 850MHz CPU, and the memory size required for Q-tables was about 5MBytes.

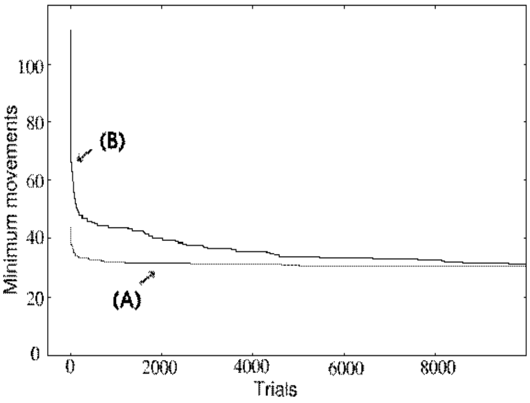

Fig. 8. Performance comparison for case 1

5. CONCLUSIONS

A new reinforcement learning system for marshaling plan at container terminals has been proposed. By using dynamic binary tree as a Q-table, the learning algorithm can be implemented for marshaling plans that have huge number of states.

In 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 assures to achieve the desired layout in each trials, the method can generate solutions with the smaller number of container movements as compared to the conventional method.

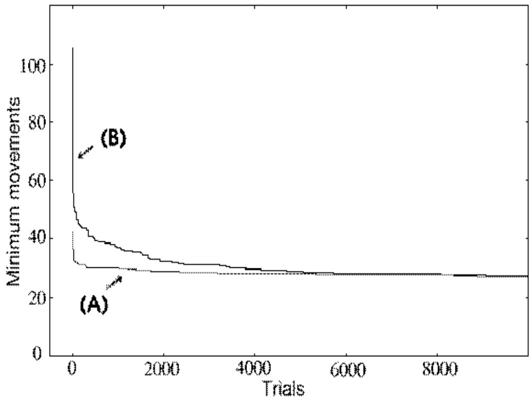

Fig. 9. Performance comparison for case 2

REFERENCES

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

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

Koza, J. R. (1992). Genetic programming on programming computers by means of natural selection and genetics.

Minagawa, M. and Y. Kakazu (1997). An approach to the block stacking problem by multi agent cooperation. *JSME Series C* **63**(608), 231–240, (in Japanese).

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

Siberholz, M. B., B. L. Golden and K. Baker (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. (1989). Learning from delayed rewards. *Ph.D. Thesis, Cambridge University, Cambridge, England*.

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