Policy Learning Using Modified Learning Vector Quantization for Reinforcement Learning Problems

Afif Mohd Faudzi, Ahmad
Department of Electrical and Electronic Engineering, Graduate School of Information and Electrical Engineering, Kyushu University | Department of Electrical and Electronic Engineering, Universiti Malaysia

村田, 純一
九州大学大学院システム情報科学研究院電気システム工学 : 教授

https://doi.org/10.15017/1560523

出版情報：九州大学大学院システム情報科学紀要. 20 (2), pp.39-44, 2015-07-24. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第20巻　第2号　平成27年7月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.20, No.2, July 2015

Policy Learning Using Modified Learning Vector Quantization for Reinforcement Learning Problems

Ahmad AFIF MOHD FAUDZI*,** and Junichi MURATA***

(Received June 25, 2015)

Abstract: Reinforcement learning (RL) enables an agent to find an optimal solution to a problem by interacting with the environment. In the previous research, Q-learning, one of the popular learning methods in RL, is used to generate a policy. From it, abstract policy is extracted by LVQ algorithm. In this paper, the aim is to train the agent to learn an optimal policy from scratch as well as to generate the abstract policy in a single operation by LVQ algorithm. When applying LVQ algorithm in a RL framework, due to an erroneous teaching signal in LVQ algorithm, the learning sometimes end up with failure or with non-optimal solution. Here, a new LVQ algorithm is proposed to overcome this problem. The new LVQ algorithm introduce, first, a regular reward that is obtained by the agent autonomously based on its behavior and second, a function that convert a regular reward to a new reward so that the learning system does not suffer from an undesirable effect by a small reward. Through these modifications, the agent is expected to find the optimal solution more efficiently.

Keywords: Policy learning, Learning Vector Quantization, Reinforcement learning and Abstraction

1. Introduction

Reinforcement learning (RL) is among the great learning frameworks that train an agent to find a solution to an unknown problem by interacting with the environment[1]. The learning framework, which is based on iterative interactions with the environment by trial-and-error, enables RL to be applied to complicated or unknown environments. However, it may take a long time to obtain the proper solution because of its trial-and-error learning framework[1]. The more complex and larger the environment is, the more exploration it requires, and it will consume more learning time or computation resources. Furthermore, if the environment changes, RL abandons past experiences and requires its agent to learn from scratch, which does not seem very intelligent nor efficient.

Many studies have been done to improve RL methods that provide skills or prior knowledge to improve an agent's interaction with the environment such as Hierarchical RL[2] and Option[3], and they have been proven to enhance the learning process. Besides that, an agent can also benefit from their own past experiences, i.e., the knowledge obtained from solving earlier problems. In the previous research, the authors applied the knowledge acquired from a previous problem to guide an agent's exploration in another similar problem. The transferred knowledge was abstracted before being applied and learning vector quantization (LVQ) was proposed to perform the abstraction[4, 5]. The abstraction was performed on a learned policy trained by Q-Learning[6]. The results showed that the abstraction using the LVQ algorithm was successful, the abstract policy represented by weight vectors was simple and easy to interpret[4] and the application of abstract policy from previous similar environment accelerated the initial exploration of the agent in a new environment[5]. In both studies, LVQ algorithm was used only as a supporting method. First, the LVQ algorithm was used to extract an abstract policy from the learned policy that was generated by Q-learning[4]. Second, during the application of abstract policy to the new problem, the LVQ algorithm was used as an action function in Q-learning learning framework[5].

The objective of this paper is to extend the LVQ algorithm to be the leading method so it can enable the agent to learn an optimal policy without conventional RL methods. If it can be realized, only a single operation is required for the preparation of abstract policy from scratch and also for an adaptation of the agent when the environment or the problem changes. The challenge in a policy learning using LVQ algorithm is, the teaching signal provided by the environment during learning might not be accurate and might mis-train the agent[7]. The learning process might end up with a non-optimal solution. In this paper, the optimal solution means the best solution i.e. agent uses the shortest path from the start state to the goal state and the op-

*Department of Electrical and Electronic Engineering, Kyushu University, Fukuoka, Japan
**Department of Electrical and Electronic Engineering, Universiti Malaysia Pahang, Malaysia
***Department of Electrical Engineering, Kyushu University, Fukuoka, Japan

timal action means the best action in each given state that bring to the optimal solution.

Abramson and Wechsler[8] proposed S[arsa]LVQ that introduced a parameter in LVQ algorithm so that the type of actions are considered during learning. A better action will receive a higher reward. Similarly, Shon et al. proposed a modified LVQ algorithm which limits the learning for non-optimal behavior or action by introducing a modified learning rate[7]. The results showed that the modified LVQ algorithm outperformed the ordinary LVQ algorithm in term of getting an optimal policy and also outperformed the Q-learning method in term of learning time. However, the limit calculation in the modified LVQ required the designer to provide an additional information e.g. ideal number of steps from the start state to the goal state.

In this paper, the authors present another method to enable the agent autonomously to find the optimal solution, without additional information except the scalar reward obtained at the end of each episode. It is realized by setting the learning limit, the same idea as Shon's proposal but the calculation is based on only the agent's experience during the learning process. A simple 2D maze problem is designed to verify the validity of the proposed algorithm. Simulation result shows that the agent successfully find the optimal path using the modified LVQ algorithm in most cases.

The rest of this paper is organized as follows. In the next section, LVQ algorithm is explained. Then, Section 3 describes the issue and the proposed solutions. In Section 4, simulation settings and result are described. Finally, Section 5 states the conclusions and future work.

2. LVQ algorithm

Learning vector quantization is a supervised learning algorithm. It is one of the appropriate algorithms to apply when a designer wants to classify a set of labeled input data[9]. As shown in **Fig. 1**, the LVQ network consists of an input layer and an output layer. These layers are connected with each other. The input layer receives an input $\boldsymbol{x} = [x_1 \dots x_n]^T$ which belongs to category T. Each of the output layer nodes has a weight vector $\boldsymbol{w}_j = [w_{j1} \dots w_{jn}]^T$ and a pre-assigned label C_j as the output. During learning, the weight vectors are trained to provide the correct labels for all input data.

The LVQ algorithm can be summarized as follows:

(1) Initialize the weight vectors $\boldsymbol{w}_j$, $j = 1, \dots, c$ of the LVQ network, where c is the total number of the output nodes.
(2) Input the input vector $\boldsymbol{x}$ to the LVQ network.
(3) Calculate distance d_j between the input vector $\boldsymbol{x}$ and weight vector $\boldsymbol{w}_j$ as in Eq. (1),

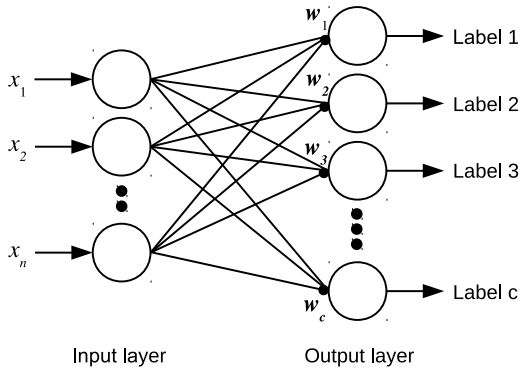


Fig. 1 Structure of LVQ network

$$d_j = \sqrt{\sum_{i=1}^{n} (x_i - w_{ji})^2}. \tag{1}$$

(4) Find the minimum among $d_j, j = 1, \dots, c$ and denote it by d_{win}.
(5) Update $\boldsymbol{w}_{win}$ as in Eq. (2),

$$\begin{aligned} \boldsymbol{w}_{win}(t+1) &= \boldsymbol{w}_{win}(t) + \alpha(t)[\boldsymbol{x}(t) - \boldsymbol{w}_{win}(t)], \\ &\quad \text{if } C_{win} = T, \\ \boldsymbol{w}_{win}(t+1) &= \boldsymbol{w}_{win}(t) - \alpha(t)[\boldsymbol{x}(t) - \boldsymbol{w}_{win}(t)], \\ &\quad \text{if } C_{win} \neq T, \end{aligned} \tag{2}$$

where α $(0 < \alpha < 1)$ is the learning rate.
(6) Go to step 2.

In steps 4 and 5, the LVQ network selects a weight vector closest to the given input vector and then compares the output's category label of LVQ network with the correct category T. If they match, the selected weight vector is updated so that it approaches the input vector. Otherwise, the chosen weight vector is updated so that it moves away from the input vector. The learning rate $\alpha(t)$ in Eq. (2) is a value that controls the distance of the adjustments made during the training process and its value usually gradually decreases along the learning process.

2.1 LVQ Variants

Improved versions of the algorithm mentioned above which also known as LVQ1 are OLVQ1, LVQ2.1, and LVQ3. OLVQ1 is the same with LVQ1, except that each weight vector has its own learning rate. In LVQ2.1, two winning neurons are selected and only updated if one belongs to the desired class and one does not, and the distance ratio is within a defined window. The LVQ3 is the same as LVQ2.1 except if both winning neurons are of the correct class, they are updated but adjusted using an epsilon value, the adjusted learning rate instead of the global learning rate[10].

3. Issue and solutions

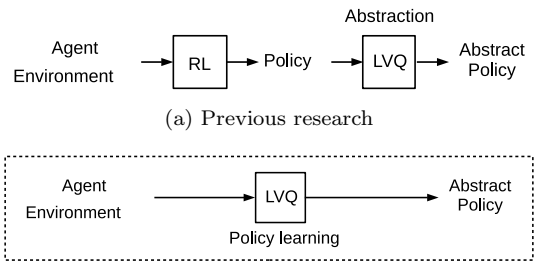
3.1 Issue

In the previous research, as shown in **Fig. 2a**, LVQ was proposed to extract an abstract policy out of learned policy that was trained by Q-learning. In this paper, the authors would like to perform the whole process including policy learning from scratch until the abstract policy generation with a single operation using LVQ algorithm (see **Fig. 2b**). If it is possible, we can also expect an adaptation using LVQ algorithm when the problem or environment changed, which is not covered in this paper.

In RL, the optimal solution is learned based on the reward that is provided by the environment. For example, in Q-learning, the evaluation value of state-action pair, Q is updated each time after an action taken by the agent. After learning, the pair that has the highest evaluation value in a state will detemine what action should be taken. On the other hand, the LVQ network's weight vectors are trained by supervised learning. In a RL framework, the LVQ network's input vector senses the state, while the label of winning node whose weight vector is the closest to the input vector decides the action. When the agent reaches the goal, all activated nodes will be rewarded i.e. approach more closer to each input vector. However, since the only condition is 'the agent reaches the goal', the learning process may end up with complete the task but with a non-optimal solution.

Imagine a simple maze task with an agent and a goal. In this task, the agent is trained to move from a fixed start state to the goal state by avoiding the wall using the shortest path. There is more that one path that can lead the agent to the goal. All actions that lead the agent to the goal will get a reward. For example, in a certain state, an action 'move forward' is selected and performed. This action eventually leads the agent to the goal and thus may received a reward i.e. the weight vector of the activated node may be updated to approach the state's input vector by LVQ algorithm. Once it is updated to be closer to the input vector, it becomes more likely to be selected in the future. However, in term of shortest path, another action 'move right' might have been the best action for the state. Since the action 'move forward' is repeatedly selected, its weight vector can be closer to the input vector the optimal action 'move right'. After learning, the agent may end up with the non-optimal policy rather than the optimal one.

The conventional LVQ algorithm must be modified so that it can learn the optimal solution properly even though the teaching information provided by the designer is limited.



(a) Previous research

(b) Current research

Fig. 2 Learning flow

3.2 Modified LVQ algorithm

In this paper, the authors propose two modifications to prevent the problem mentioned above so that the network's weight vectors are updated efficiently and that the agent can learn the optimal solution.

The first modification is regarding the reward parameter. As shown in Eq. (2), the conventional LVQ algorithm does not have a reward parameter. Shon et al. have improved this in their proposed modified learning algorithm that can be expressed by Eq. (3-4)

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k S_k r_k (\boldsymbol{x}_k - \boldsymbol{w}_k), \tag{3}$$

where k is the learning episode number, $\alpha_k (0 < \alpha_k < 1)$ is the learning rate,

$$S_k : \begin{cases} +1 & \text{if reward is received,} \\ -1 & \text{if punishment is received.} \end{cases}$$

r_k is the amount of reward

$$r_k = \frac{1}{total\_step_k - \beta}, \tag{4}$$

where $total\_step_k$ is the number of moves that the agent has made in the episode k and β is the parameter that adjusts the relation between r_k and $total\_step_k$[7]. Unfortunately, there are two possible drawbacks. The first is, that it is required the designer to provide the parameter β and the second is, that if the start state or the goal state is changed, the parameter β also need to be modified.

Here, a new calculation for reward is proposed. It can be expressed as Eq. (5)

$$r_t = \frac{1}{total\_step_k - t} r_{max}, \quad (t = 0, 1, ..., total\_step_k - 1), \tag{5}$$

where t is the time step and r_{max} is the parameter that adjusts the reward. Through this calculation, the activated node in the first time step will receive a reward as

low as $r_0 = r_{max}/total\_step_k$ and in the last time step will receive a reward as high as $r_{total\_step_k-1} = r_{max}$. Thus, the shorter path with a smaller $total\_step_k$ will provide a higher reward and moves the selected node more closely to the input vector.

The second modification presents a limit on how a weight vector can approach the input vector, which is not provided by the conventional LVQ algorithm. The learning limit is required to prevent the unwanted weight vector, which leads to the non-optimal action, to approach the input vector. In the conventional LVQ algorithm, only the distance between the input vector and the weight vectors matters. However, when we put in RL framework, the importance of received reward value should also be highlighted.

Several reward functions for a new reward $R_{t,s}$ that depends on a reward r_t between the minimum and the maximum reward the agent ever received for a certain state s is shown in **Fig. 3**. If we update the weight vectors using a reward $R_{t,s} = r_t$, as shown in **Fig. 3**, all the activated nodes including that receive a small reward, or in other words, that provide non-optimal actions also get rewarded, which is unnecessary. However, if we update the activated node that receives only a highest reward as shown in **Fig. 3**($R_{t,s} = f_{ideal}(r_t)$), there is a possibility that the system ignores some nodes that might have been the best action for certain states. They received a low reward perhaps due to the non-optimal path ahead of them. Therefore, a reward function as shown in **Fig. 3**($R_{t,s} = \alpha_{r_t,s} r_t$) that can be expressed as Eq. (6) is proposed.

$$\begin{aligned} \alpha_{r_t,s} r_t &= \frac{r_t - r_{min,s}}{r_{max,s} - r_{min,s}} r_t, \\ &= \frac{1}{r_{max,s} - r_{min,s}} {r_t}^2 \\ &\quad - \frac{r_{min,s}}{r_{max,s} - r_{min,s}} r_t, \end{aligned} \tag{6}$$

where $r_{min,s}$ and $r_{max,s}$ is the minimum and the maximum rewards respectively that the agent ever receives for a certain input state during the learning process. Through Eq. (6), the learning for an action that received a small reward can be limited, which provides an opportunity to the action that received a high reward to approach the input vector.

Compared to our modified LVQ algorithm, LVQ1 does not have a reward parameter and LVQ2.1 only updated if one of two selected neurons belongs to the desired class and one does not, and the distance ratio is within a defined window, which is not suitable for RL framework where more than one neurons may lead the agent to the goal state. On the other hand, the proposed algorithm more closed to LVQ3 that treats neighborhood neurons

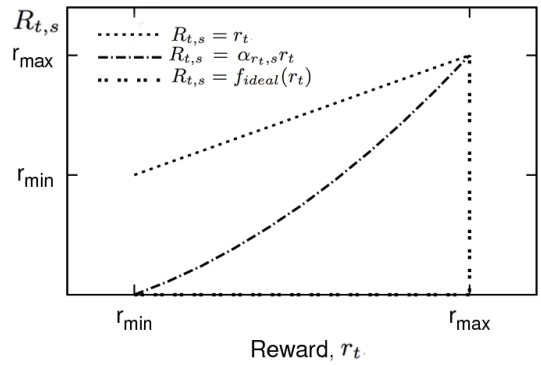


Fig. 3 A new reward value $R_{t,s}$

with less restrictive adaptation rules, except it just like other former LVQ algorithms that considers only the distance which is not enough for RL framework.

4. Simulation

In order to verify the validity of the proposed method, a two-dimension maze problem is treated. It is a 20×20 maze. In the maze, an agent is trained to find the shortest path from the start state to the goal state. As shown in **Fig. 4**, both start state and goal state are fixed.

As illustrated in **Fig. 5a**, in each state s, the agent senses its present position and the existence of obstacles in its neighborhood. Therefore, each input vector consists of the position of the agent(x_1, x_2), and the existence of obstacles at its front(x_3), right(x_4), back(x_5) and left(x_6) cells. As shown in **Fig. 5b**, the agent can perform four actions a which are 'move forward', 'move right', 'move back' and 'move left'. In every time step t, the agent can make only one action. If the agent takes an action towards the obstacles, the time step is counted, however, the agent stays in the same state.

In this task, the agent will start from the start state and tries to reach the goal state by avoiding the obstacles. During learning, the agent decides its actions based on Boltzmann selection. As shown in **Fig. 6**, the temperature for Boltzmann selection starts from 0.15 and decreases to 0.001 as the learning proceeds. The high temperature will enable the agent to perform an exploration by providing a random action instead of winner node's action. When the temperature decreases, greedy action will be performed.

An episode is counted from the instant that the agent sets off the start state until the moment that it reaches a terminal state which is the state where the agent reaches the goal state or when the agent in the time-out state. At the end of every episode, the weight vectors are updated. When the agent successfully reaches the goal, the weight vectors of the output nodes activated along the path that moves the agent from the start state to

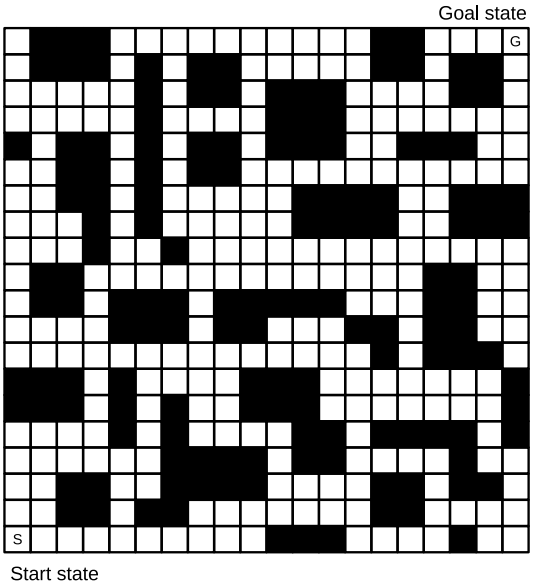


Fig. 4 Maze used in the simulation. The start state and the goal state are fixed.

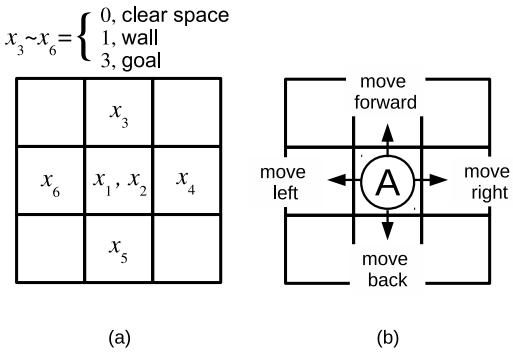


Fig. 5 (a) Components of input vector; an input vector consists of the position of the agent(x_1, x_2), and the existence of obstacles at its front(x_3), right(x_4), back(x_5) and left(x_6) cells. (b) Actions of the agent; the agent can perform four actions which are 'move forward', 'move right', 'move back' and 'move left'.

the goal state are updated by Eq. (7) and Eq. (8)

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k R_{t,s} S(\boldsymbol{x}_k - \boldsymbol{w}_k), \tag{7}$$

$$S = 1, \tag{8}$$

where k is the episode number and $R_{t,s}$ is the proposed reward function as stated in Eq. (6). The learning rate α_k is calculated based on the optimized-learning-rate LVQ1 (OLVQ1) that can be expressed by the Eq. (9)

$$\alpha_k = \frac{\alpha_{k-1}}{1 + s\alpha_{k-1}}, \tag{9}$$

with $\alpha_0 = 0.001$. The maximum reward r_{max} in Eq. (5) is set as 100. On the other hand, when the agent fails to touch the goal within the time step limit, which is set

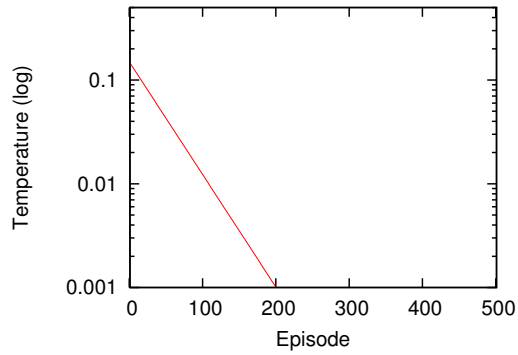


Fig. 6 Temperature cooling schedule that is used in the action selection.

as 2000 steps, or when the activated output node tries to move the agent toward the obstacle, the reward $R_{t,s}$ and the parameter S are set as

$$\begin{aligned} R_{t,s} &= \quad 0.05, \\ S &= \quad -1. \end{aligned} \tag{10}$$

Training was done for 300 episodes with 10 different randomly initialized network's weights. Each state has four weight vectors close to its input vector indicate the four available actions.

As result, **Table 1** shows the number of steps that was obtained through the conventional Kohonen's LVQ algorithm and the modified LVQ algorithm after the simulation completed. In the simulation of the conventional LVQ algorithm, the reward $R_{t,s}$ was set as 1.0. As shown in **Table 1**, the conventional LVQ algorithm hardly found the path from the start state to the goal state. The agent managed to complete the task in three out of ten cases, but found the optimal path just in a single case. On the other hand, the modified LVQ algorithm successfully completed the task in all cases and found the optimal path in eight out of ten cases.

Even though the optimal solution was obtained, the abstract policy was not. Each input state has its own

Table 1 The number of steps of the best path after the simulation completed.

Random number sequence	Conventional LVQ algorithm	Proposed LVQ algorithm
1	Failed	38
2	Failed	38
3	Failed	40
4	40	38
5	Failed	38
6	38	38
7	46	38
8	Failed	42
9	Failed	38
10	Failed	38

weight vectors. However, if we want to apply the learned policy to another similar task, only 38 activated node's weight vectors are need to be transferred instead of the whole table lookup that usually used in Q-learning.

5. Conclusions

This paper has proposed two modifications of the Kohonen's LVQ algorithm so it can work well in reinforcement learning framework i.e. learning an optimal policy. These modifications also overcome the drawbacks arose in previous similar research. The modified LVQ algorithm introduces, first, a regular reward that is obtained by the agent autonomously based on its behavior, and second, a function that convert a regular reward to a new reward so that the learning system does not suffer from an undesirable effect by a small reward. The new reward realized that by limits the distance that a weight vector can approach to an input vector. The simulation result showed that the modified LVQ algorithm performed better in terms of successfully completed the given task and found the optimal solution in the most cases. A future study investigating on how to acquire not only the optimal solution but also the abstract policy directly would be interesting.

References

1) R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction", *MIT Press*, 1998.
2) Yuchen Fu, Quan Liu, Xionghong Ling, and Zhiming Cui, "A Reward Optimization Method Based on Action Subrewards in Hierarchical Reinforcement Learning", *The Scientific World Journal*, vol. 2014, Article ID 120760, 6 pages, 2014.
3) Terashima, K., Takano, H., and Murata, J., "Acceleration of Reinforcement Learning with Incomplete Prior Information", *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 17(5), 721-730, 2013.
4) A. A. M. Faudzi, H. Takano, and J. Murata, "A study on visual abstraction for reinforcement learning problem using Learning Vector Quantization", *SICE Annual Conference (SICE), 2013 Proceedings of* , pp. 1326-1331, 2013.
5) A. A. M. Faudzi, H. Takano, and J. Murata, "A study on abstract policy for acceleration of reinforcement learning", *SICE Annual Conference (SICE), 2014 Proceedings of* , pp. 1793-1798, 2014.
6) Watkins, C. J., and Dayan, P., "Q-learning", *Machine Learning*, Vol. 8, pp. 279-292, 1992.
7) Kyu, S. M., Murata, J., and Hirasawa, K., "Behavior Learning of Autonomous Robots by Modified Learning Vector Quantization", *Transactions of the Society of Instrument and Control Engineers*, 37, 1162-1168, 2001.
8) Abramson, M., and Wechsler, H., "Competitive reinforcement learning for combinatorial problems", *In Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 4, pp. 2333-2338. IEEE, 2001.
9) Kohonen, T., "Learning Vector Quantization in Self-Organizing Maps SE - 6", *Springer Berlin Heidelberg*, Vol. 30, pp. 245-261, 2001.
10) Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., and Torkkola, K., "Lvq pak learning vector quantization program package", *Technical report, Laboratory of Computer and Information Science Rakentajanaukio 2 C*, 1991-1992, 1996.