Transfer Learning Through Policy Abstraction Using Learning Vector Quantization

Ahmad Afif Mohd Faudzi[1], Hirotaka Takano[2] and Junichi Murata[3]
[1]*Faculty of Electric and Electronics Engineering, Universiti Malaysia Pahang, Pekan, Pahang, Malaysia.*
[2]*Department of Electrical and Electronics Engineering, Faculty of Engineering, University of Fukui, Fukui, Japan.*
[3]*Department of Electrical Engineering, Kyushu University, Fukuoka, Japan.*
afif@ump.edu.my

***Abstract*—Reinforcement learning (RL) enables an agent to find a solution to a problem by interacting with the environment. However, the learning process always starts from scratch and possibly takes a long time. Here, knowledge transfer between tasks is considered. In this paper, we argue that an abstraction can improve the transfer learning. Modified learning vector quantization (LVQ) that can manipulate its network weights is proposed to perform an abstraction, an adaptation and a precaution. At first, the abstraction is performed by extracting an abstract policy out of a learned policy which is acquired through conventional RL method, Q-learning. The abstract policy then is used in a new task as prior information. Here, the adaptation or policy learning as well as new task's abstract policy generating are performed using only a single operation. Simulation results show that the representation of acquired abstract policy is interpretable, that the modified LVQ successfully performs policy learning as well as generates abstract policy and that the application of generalized common abstract policy produces better results by more effectively guiding the agent when learning a new task.**

***Index Terms*—Abstraction; Learning Vector Quantization; Reinforcement Learning; Transfer Learning.**

I. INTRODUCTION

Reinforcement learning (RL) is among the great learning frameworks that trains an agent to find a solution to a 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, because it is based on exploration, it may also take a long time to obtain the proper solution [2-3]. 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 abandon 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 Option [2] and Hierarchical RL [4], 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. Recent studies show that previous knowledge acquired from different but related problems can guide the agent better during the exploration of new environments, which is also known as transfer learning [5]. However, the fact that we do not know for sure whether the obtained knowledge may or may not work in unknown different environments is still needs to be considered [6-7]. In this paper, the authors improve the transfer learning by considering an abstraction, which is expected to help the designer by its simplifying ability.

Abstraction is an operation that changes the representation of an object by removing less critical details while preserving desirable properties [8]. Rajendran and Bergamo proposed abstract policy learning and reused the abstract policy to improve initial performance of an RL learner in a similar new problem [9-10]. They showed good results in terms of the learning acceleration. However, they did not consider any other environments.

In this research, the type of knowledge that is transferred between tasks is a policy. Q-learning method is used as based learning method and a modified learning vector quantization (LVQ) is proposed [11-13]. The proposed method considers not only the state values but also the action to be taken. In [11], the abstraction was performed on learned policy. The result showed that the abstraction was successful and the abstract policies represented by weight vectors were simple and easy to interpret. Furthermore in [12], the abstract policies obtained from previous similar environment were used to guide the initial exploration of the agent in a new environment. The result showed that the application of abstract policy from previous environment accelerated the learning in a new environment.

In this paper, the authors proposed an adaptation process by extended the LVQ algorithm to be the leading player. Through the modified algorithm, the learning system enable the agent to reuse the previous task's abstract policies instantly without the requirement of any special operation. It is also expected to train the agent to adapt to the trained environment as well as generate a new abstract policy in a single operation. As the precaution for future unknown tasks, here, a common abstract policy, which is an extracted abstract policy from the similarities of two or more environments is introduced.

A 3-D maze problem with a camera-mounted agent and several environments are considered in simulations. The results show that the agent manages to leverage the previous task's abstract policy for policy learning as well as directly generates abstract policies in a new environment using only proposed modified LVQ algorithm. The authors also find that the use of a common abstract policy presents better results than the use of policy or abstract policy from a specific environment.

The rest of this paper is organized as follows. In the next section, two main algorithms that are used in this paper are explained. Then, Section 3 describes the issues and the

proposed solutions. It will be followed by Section 4 explaining the methodology. In Section 5, simulation settings and results are described. Finally, Section 6 states the conclusions and future work.

II. Reinforcement Learning and Learning Vector Quantization

A. Reinforcement Learning and Q-learning

Reinforcement learning is a policy discovery through trial-and-error exploration [1]. The learner has to interact with the environment and discover which actions return the highest rewards by performing them. The requirements are simple. The learner needs a goal, capable of sensing the state of the environment and able to take actions that affect the state. One of the commonly used methods in RL is Q-learning.

In Q-learning, state-action pairs are evaluated, and the evaluation value is called Q-value [6]. At each step of time, an agent observes the vector of state s_t, then chooses and applies an action a_t. As the process moves to state s_{t+1}, the agent receives a reward or punishment r_{t+1}. The goal of the training is to find the sequential order of actions, which maximizes the sum of the future reinforcements. The transition rule of Q-learning is a very simple formula:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a)) \quad (1)$$

where γ is a discount factor ($0 \leq \gamma < 1$) and α is the learning rate ($0 < \alpha < 1$). In this paper, the action a_t is selected based on ε-greedy. ε is reduced according to the progress of learning using an exponential function as the next equation.

$$\varepsilon = e^{(-episode \times decay)} \quad (2)$$

where the *episode* is the current episode number, and *decay* is a parameter that determines the curve shape of the exponential function. With a probability of ε, the action is chosen randomly, otherwise, greedy action selection is done, i.e. the action with the maximum Q-value is selected.

B. Learning Vector Quantization

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 [14]. As shown in Figure 1, the LVQ network consists of an input layer and an output layer. These layers are connected with each other. The input layer will receive an input $\boldsymbol{x} = [x_1 \cdots x_n]^T$ which belongs to category T. Each of the output layer nodes has a weight vector $\boldsymbol{w}_j = [w_{j1} \cdots w_{jn}]^T$ and a preassigned 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 $\boldsymbol{w}_j, j = 1, \cdots, c$ 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 Equation (3),

$$d_j = \sqrt{\sum_{i=1}^{n} (x_i - w_{ji})^2}. \quad (3)$$

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

$$\begin{aligned} \mathbf{w}_{win}(t+1) &= \mathbf{w}_{win}(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{w}_{win}(t)], \\ &\text{if } C_{win} = T, \\ \mathbf{w}_{win}(t+1) &= \mathbf{w}_{win}(t) - \alpha(t)[\mathbf{x}(t) - \mathbf{w}_{win}(t)], \\ &\text{if } C_{win} \neq T, \end{aligned} \quad (4)$$

where α is the learning rate ($0 < \alpha < 1$).

6. Go to step 2.

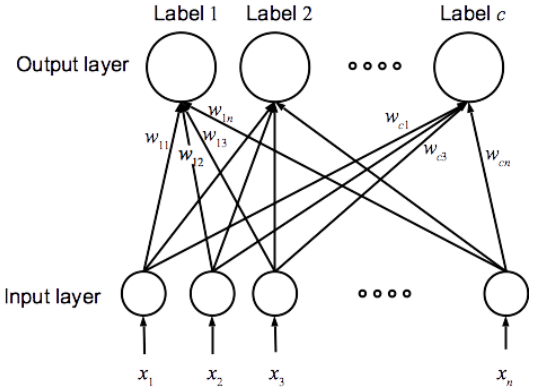

Figure 1: Structure of LVQ network

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.

There are at least three important parameters that the the designer needs to decide for LVQ algorithm. The first one is how many weight vectors should be used, and the second is where the weight vectors should be initialized or what the initial values of weight vectors should be. The last one is how the learning process will be terminated. In this paper, LVQ algorithm is modified and used as an abstraction, an adaptation and a common abstract policy generation method.

III. Issues and Solutions

Reinforcement learning enables the agent to learn a proper behavior through the trial-and-error mechanism, which is required in order to find the best actions that will return the highest rewards. However, the trial-and-error mechanism or the exploration may make the learning took a long time to provide the proper solution. Furthermore, when the task changes, RL requires its agent to learn from scratch, which does not seem very intelligent nor efficient. Here, the authors wish to alleviate the problem by transfer learning or specifically by taking advantage of the obtained knowledge from previous similar tasks.

There are several issues possibly arise when we want to perform transfer learning. The first issue is, even the previous task's policy is obtained, and we do not know for sure that it

may work in the current task. The policy might be incorrect, and its representation might not be so appropriate to be reused. If the old environment is small, and we use a lookup table to represent the policy, perhaps we manage to interpret and understand the learned policy. However, when the environment is large and complicated, it will be difficult. On the other hand, the second issue that might arise is, there is also the possibility that the current or the new tasks are unknown or unpredicted. The transferred knowledge might not be perfect for the new tasks that are unknown or unpredicted.

As shown in Figure 2 to treat the first issue, we proposed to extract an abstract policy out of the learned policy. The abstraction is expected to provide a simple and general representation that is interpretable so that we can understand the obtained policy. In this paper, we assume that some of the similar states in the learned policy might correspond the same actions. Due to that assumption, the abstraction is done by classification of the state-action pairs into a small number of groups based on the continuity in the state space and paired actions. In order to realize that, LVQ algorithm, which is an appropriate algorithm for classifying a set of labeled input data is proposed as an abstraction method. In the previous research, the results showed that the representation of abstract policy was interpretable, and the reuse of previous task abstract policy to guide the exploration successfully accelerated the agent's learning [11-12].

There are two possible approaches that can be considered when having unknown or unpredicted tasks. The first approach is an adaptation, which the system changes itself so that it can work well after the environment has actually changed. The other is to prepare the system so that it can work in all possible environments before the changes actually occur. In this paper, we consider both approaches and propose LVQ algorithm to realize them. For the adaptation, the conventional RL method may enable the agent to adapt to the environment. However, if we use the conventional RL, the obtained knowledge that was extracted in an abstract form cannot directly be used. Some special operations may require in order to take the advantage of the obtained knowledge.

Furthermore, since the RL method only produces a policy, another abstraction process is required to generate a new abstract policy for a new task. On the other hand, as shown in Figure 2, the proposed LVQ algorithm can directly use the obtained knowledge and not only trains the agent to adapt to the environment or to learn the policy but also directly generate the new abstract policy in a single operation. For the preparation of the learning system before the environment actually changes, the authors introduce another type of prior information named common abstract policy. The advantage of the common abstract policy compared to a policy or an abstract policy that obtained from a single task is it has a higher generality, thus can support the learning agent better.

IV. METHODOLOGY

In this section, we explain how the proposed treatments work using an example and also about the detailed procedures of treatments.

A. Abstraction

Imagine a simple maze task built with an agent, a goal and several types of obstacles, e.g. rocks and trees. In this task, the agent is trained to move from an initial position to the goal by avoiding those obstacles using the shortest path. After the training by reinforcement learning has been completed, the agent acquired an optimal policy that guided it to move towards the goal by turning away from the obstacles. This policy is represented by a set of input data corresponding to each of

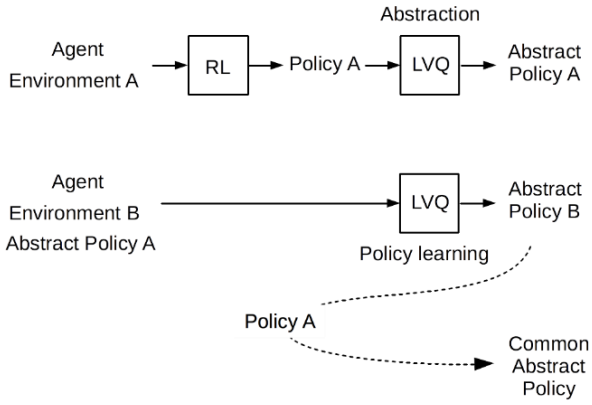

Figure 2: The learning flow. First, an abstract policy is extracted from the learned policy. Second, the obtained abstract policy is used in another similar task, which the agent requires to learn policy and generate a new abstract policy. Finally, the generation of common abstract policy that generalizes the previous abstract policies.

the several actions. In other words, the input data, which hold all states' information, are classified into several action classes.

The abstraction is performed using LVQ algorithm by classifying the input data in each action class into several subclasses. For example, the input data belonging to the turn-left class may be classified into two subclasses: one corresponds to the situation where the agent faces the tree and the other subclass corresponds to the situation where the agent faces the rock. The abstraction involves both of supervised and unsupervised learning. Since the correct class that each of the input data vectors belongs to is known, the supervised learning is to be used. On the other hand, we do not know which subclass each of them belongs to, and therefore, unsupervised learning must be performed at the same time.

Each subclass is represented by a weight vector. After the learning completed, the weight vectors found by LVQ learning serve as representative data vectors. These representative data vectors are useful for the interpretation of the subclasses by human designers and will also be practical to classify the new data whose class and subclasses are unknown. In this research, these representative data vectors are used as an abstract policy. The abstract policy is expected to provide the same performance as the regular policy but has a fewer data. It is more efficient to perform transfer learning using the lesser data with the same performance. Furthermore, since the abstract policy is consisted by the representative data vectors, it might be useful for the agent to determine an action for unknown states in new tasks.

As mentioned, the abstraction process will classify the input data into subclasses which are represented by weight vectors or nodes in LVQ algorithm. The information of the correct action will guide the learning system to classify correctly. However, which subclass each of the input data belongs to and how many nodes are required to classify all input data are unknown. Therefore, to define the number of nodes at the output layer beforehand which are required in the

original algorithm is difficult. Less number of nodes may end up with some unclassified input data and an over a number of nodes may end up with an inefficient result.

In order to overcome the above difficulty and to be functioning as an abstraction method, we proposed a slight modification of the original LVQ algorithm. In this paper, the number of nodes is dynamically changed. In the first episode, each class only has one node or one subclass. This number of nodes will be changed at the end of every episode until all the input data have been classified.

B. Policy Learning

Next, the agent is placed in a different but similar environment. The task is the same as the previous one and the obtained abstract policy from the previous task is provided. In the previous research [13], the transferred abstract policy was used as a guidance of exploration during the adaptation to a new environment through RL. In this paper, the agent adaptation is trained using LVQ algorithm and the transferred abstract policy is applied as agent's prior information. Instead of guiding the agent's exploration, the abstract policy is expected to improve the early stage of agent's exploitation. Since LVQ is used, the transferred abstract policy can directly use as the LVQ network's weight vectors, and after the learning completed, a new abstract policy can be expected.

LVQ algorithm originally proposed for supervised learning. Here, however, there are no training data provided. During learning, in each state, the agent will perform the action associated with the winner node for the input vector and it cannot be known which the correct action is. Here, the modified LVQ will update only the weight vectors of selected actions that receive the reward. In addition, there are three operations involving the weight vectors, namely; weight vector movement, weight vector addition and weight vector deletion. The weight vector movement operation will move the existed weight vectors to maximize the rewards. The weight vector addition operation will add new weight vector when there are still input vectors that failed to be classified after several times of trial. Finally, the third operation will remove the weight vectors that are not being used to prevent them from affecting other classifications.

The modified LVQ algorithm for the weight vector movement operation can be expressed by Equation (4).

C. Policy Learning

After the agent is trained in two similar but different tasks, two different policies are obtained. Since both environments are different but similar, there are three cases that can be assumed. The first case where both policies provide the same action for the same state in each task, i.e. the policy that guides the agent to move forward when the agent is one step in front of the goal. The second case is when the policies provide different actions for the same state in each task, e.g. when the agent is facing the same obstacles, but both tasks have the different optimal route solutions that require the agent move to different ways. The last case is when both policies only work on their own task. In this paper, the authors generate another abstract policy named as `common abstract policy' by extracting the similarities from both policies e.g. the first case policies. Since all task-specific policies that might provide the wrong action for the unknown task are not included, it is expected to be more reliable compared to previous task specific's abstract policies.

The common abstract policy is generated using the LVQ algorithm. The procedure is quite simple and it requires only an abstract policy from one of the past-learned environments and a set of trained data that also represents the regular policy from other past environments. The procedure can be summaries as follows:

Only the weight vectors from the nodes that have only `True' flags or both flags are selected to generate common abstract policy.

V. SIMULATIONS

In order to verify the validity of the proposed methods, maze problems with some obstacles were designed. As shown in Figure 3, there are three environments used in this paper. They are three-dimension grid environments and were built using the Google SketchUp software. The locations of the goal and the obstacles (water and boxes) in each environment were designed differently. In this paper, the agent's task is to avoid the obstacles while finding the shortest route from the initial positions to the goal.

The agent has four possible states *s* in each coordination; facing north, east, west and south. As illustrated in Figure 4(a) and Figure 4(b), each state sensed by an image data captured by the camera, the distance between itself with the goal and the direction of the goal. Before the camera image is used as a part of state information, it is pre-processed to reduce the resolution to 4×3 pixels. There are 38 signals for each state; 36 signals from the captured image, one signal that indicates the distance between the agent and the goal and one signal for indicating the direction of the goal. As shown in Figure 4(c), in every time step *t*, the agent can make an action *a* out of three possible actions, namely, ‘move forward in one grid’, ‘turn right’, and ‘turn left’. If the agent takes an action towards the obstacles, the time step is counted, however, the agent stays in the same state.

There are three stages of learning; an abstraction, an adaptation and common abstract policy generation.

A. Abstraction

In the first stage, before the simulation of abstract policy acquisition is done, the trained policy is generated. For that, the agent is trained in the environment A and Q-learning method is used. After the learning was completed, for the environment A, the learned policy consisting of 275 state-action data was generated. Figure 5 shows a part of the learned policy that corresponds ‘turn right’ action. Next, the acquisition of abstract policy is performed using the proposed method, LVQ algorithm. The detailed parameters’ settings and the result can be referred at [11].

B. Adaptation

In the second stage, the agent is placed in a different but similar environment named ‘Environment B’ as shown in Figure 3(b). The task has still remained the same where the agent needs to find the shortest path from an initial position to the goal.

As shown in Figure 2, for the second stage only LVQ algorithm is used for policy learning as well as abstract policy generating. During the initialization process, the weight vectors that represent the ‘Abstract Policy A’ are used as the LVQ network’s initial weight vectors. The weight vectors then are updated after every episode finished. An episode is a period between the agent sets off the initial position until ends in a terminal state which is either the agent touches the goal

or the agent in time-out state. When the agent successfully touches the goal, the weight vectors of the LVQ output nodes activated along the path to the goal are updated by Equation (6-8). The learning rate $\alpha_{(k)}$ is calculated by Equation (5) with $\alpha_{(0)} = 0.001$.

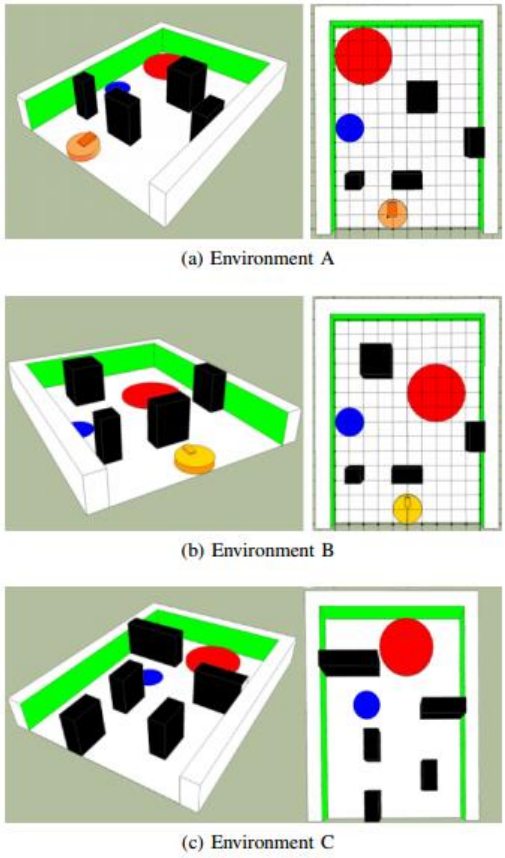

Figure 3: Task environments. There is a goal (red) and two obstacles, water (blue) and boxes (black). Locations of the goal and a box are different for both environments.

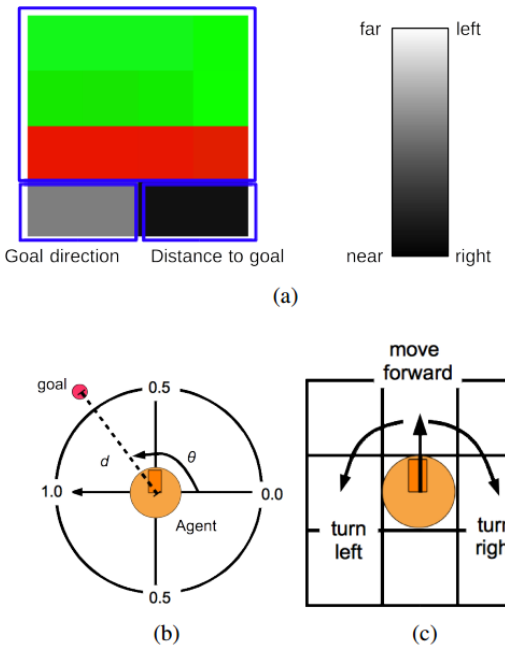

Figure 4: (a) The illustration of a state, 36 signals data from a three channels 4×3 pixels RGB images that were pre-processed before used with the two additional signals. (b) Two additional signals for the input vector; the direction of the goal θ and the distance between the agent and the goal d. (c) Actions of the agent.

The value of the reward, $r_{(k)}(t)$ depends on the agent time step *Step*(t) and r_{max} which is set as 100. *TotalStep* is the number of moves that the agent has made in episode k. When the agent failed to touch the goal within the step limit, which is set as 500 steps, the reward $r_{(k)}(t)$ and its sign s are set as 0.05 and -1 respectively.

Table 1 compares the results obtained from the simulation that was done using the modified LVQ algorithm to the results of a simulation that was done using both Q-learning and LVQ algorithm. It is apparent from this table that, compared to the former method, the number of abstract policy generated using only modified LVQ is almost double and the accumulative steps are 341 steps higher. These results show that through the proposed method, the agent successfully learned to complete the task regardless where it was initialized. However, the results also indicate that the agent could not find the optimal solutions for all initial positions. On the other hand, there are 275 different initial positions and 255 different states in 'Environment B'. Considering that, the average of exceeding steps for each start positions is less than two steps, which is acceptable. The result also shows that after learning, 255 input states were successfully classified into 174 subclasses or abstract policies.

Table 1
The accumulative steps from all initial position to the goal

Learning method	The number of weight vector	Accumulative step
Modified LVQ	174	1924
Q-learning and LVQ	91	1583

C. Common Abstract Policy Generation

In the final stage, the abstract policy that is used for LVQ network's weight vectors initialization is taken from 'Environment B' and a set of trained data that is inputted into the LVQ network is taken from 'Environment A'. After the first and the second stages finished, as shown in Table 2, there are 275 input vectors from the 'Environment A' and 174 weight vectors in 'Abstract policy B'.

As a result, Table 2 shows the number of weight vectors in 'Abstract Policy B' that provide only the correct outputs, only the wrong outputs, both correct and wrong outputs and also the number of weight vectors that were not even chosen at all. As shown in Table 2, there are 44 weight vectors that outputted only correct actions, which will be used as the common abstract policy. The common abstract policy then is tested in both environments.

Table 2
Common abstract policy generation

Training Data	Input vector from Environment A	275
	Weight vector from Abstract Policy B	174
Result: Weight vector in Abstract Policy B that outputted	Correct output only	44
	Wrong output only	99
	Correct and Wrong Output	25
	Nothing	56

Figure 5 shows the agent states in both environments that corresponded to three subclasses of the generated common abstract policy, namely Subclass 11, Subclass 19 and Subclass 31. As shown in Figure 8, we can see that for both environments, the Subclass 11 corresponded when the agent is facing the goal while the Subclass 19 and the Subclass 31 corresponded when the agent is facing the south and the green wall respectively.

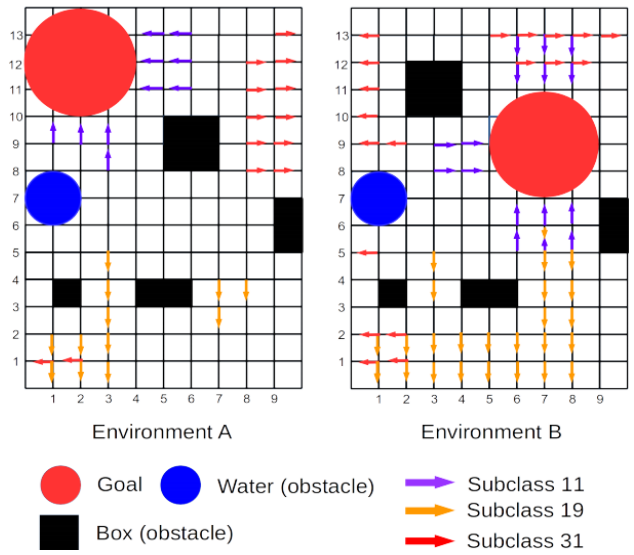

Figure 5: The agent states on both 'Environment A' and 'Environment B' that corresponded to the generated common abstract policy.

Figure 6 shows the results of the performance of Q-learning with the common abstract policy, compared to Q-learning without past knowledge, and Q-learning with 'Abstract Policy A' and 'Abstract Policy B' in environment C (Figure 4(c)). First of all, we can observe that Q-learning with past knowledge does present a better performance compared to the performance of an agent learning from scratch without using any kind of previous knowledge. In the earlier stage, the error is lower when the agent reuses previous knowledge. The application of the learned policy guides the agent exploration and thus led it to reach the goal faster. Without reuse, on the other hand, the agent takes more time to explore the environment. Then, if we compare the usage of common abstract policy to the usage of abstract policy from specific environments, a better performance of the usage of common abstract policy can be noticed, considering the performance at the final stage. This is especially due to the fact that the abstract policy from specific environments contains some policy that is environment specific. This abstract policy is not helpful and may even be disadvantageous to the agent.

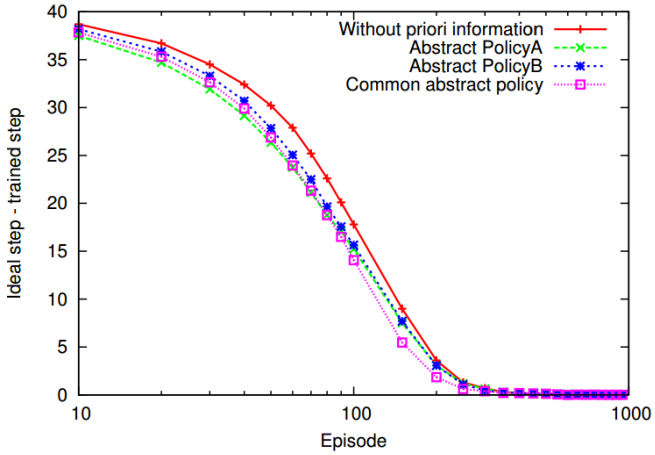

Figure 6: Performance results. Performance of Q-learning with the common abstract policy, compared to Q-learning without past knowledge, and Q-learning with Abstract Policy A and Abstract Policy B in a new Environment C shown in Figure 4(c). Each point represents the average error value of 100 executions.

VI. Conclusion

This paper has investigated abstraction in order to improve the knowledge transfer between tasks. The type of knowledge that has been considered in this study is policy. In the first stage, LVQ algorithm was proposed as an abstraction method that was performed on learned policy and abstract policy was acquired. The result showed that the representation of abstract policy was interpretable. In the second stage, the agent was placed in a new similar task and LVQ algorithm was used for policy learning as well as abstract policy generation. The learning was successful in terms of the agent capability to move toward the goal regardless where it was initialized. However, the agent could not obtain the optimal solution for all the positions. In the final stage, after two simulations in different environments, the common abstract policy was generated. The common abstract policy generalizes the similarities between two environments' policies and was tested in the third environment. The result showed that Q-learning that was guided by common abstract policy performed better compared to the others. A future study investigating more tasks and how to acquire the optimal solutions using only LVQ algorithm would be interesting.

Acknowledgment

The research was supported by Research Grant RDU1703140 from Research and Innovation Department, Universiti Malaysia Pahang. Special thanks to all members of Murata Laboratory, Kyushu University for all supports.

References

[1] R. S. Sutton and A. G. Barto, "*Reinforcement learning: An introduction*", MIT Press, 1998.

[2] 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.

[3] Koga, M. L., Freire, V., and Costa, A. H. R., "Stochastic abstract policies: generalizing knowledge to improve reinforcement learning", *IEEE Transactions on Cybernetics*, 45(1), 77-88, 2015.

[4] 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, pp. 1-6, 2014.

[5] Taylor, M. E., and Stone, P., "Transfer Learning for Reinforcement Learning Domains: A Survey", *Journal of Machine Learning Research*, 10, 1633-1685, 2009.

[6] Watkins, C. J., and Dayan, P., "Q-learning", *Machine Learning*, Vol. 8, pp. 279-292, 1992.

[7] Carroll, J. L., and Peterson, T., "Fixed vs. dynamic sub-transfer in reinforcement learning", In *ICMLA*, pp. 3-8, 2002.

[8] Ponsen, M., Taylor, M., and Tuyls, K., "Abstraction and generalization in reinforcement learning: A summary and framework", *Adaptive and Learning Agents*, 2010, pp. 1-32.

[9] Rajendran, S., and Huber, M., "Learning to generalize and reuse skills using approximate partial policy homomorphisms", In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pp.2239-2244.

[10] Y. Bergamo, T. Matos, V. da Silva, and A. Costa, "Accelerating reinforcement learning by reusing abstract policies", *VIII Encontro Nacional de Inteligncia Artificial (ENIA 2011)*, 2011, pp. 1793-1798.

[11] 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.

[12] 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.

[13] T. Kohonen, "*Learning Vector Quantization in Self-Organizing Maps SE - 6*", Springer Berlin Heidelberg, Vol. 30, pp. 245-261, 2001.