Rule Abstraction and Transfer in Reinforcement Learning by Decision Tree

Min Wu, Atsushi Yamashita and Hajime Asama

***Abstract*— Reinforcement learning agents store their knowledge such as state-action value in look-up tables. However, loop-up table requires large memory space when number of states become large. Learning from look-up table is tabula-rasa therefore is very slow. To overcome this disadvantage, generalization methods are used to abstract knowledge. In this paper, decision tree technology is used to enable the agent to represent abstract knowledge in rule from during learning progress and form rule base for each individual task.**

I. INTRODUCTION

Motion planning requires a mobile robot to generate collision-free trajectories from its initial point to desired destinations during moving in environment. However, to compute the whole trajectory is infeasible in real time application when the environment is unknown to the agent. Reinforcement learning (RL) [1], [2] is suitable in this situation as RL enables agents to learn correct policy by trial-and-error interactions with dynamical environment.

However, in RL, learning agent starts trail-and-error progress without knowing anything about the environment. Therefore the agents should repeat trying every possible action to ensure correct learning result in every new task and result in bad initial performance. Secondly, as the number of states and possible actions growing, the size of state-space as well as computation power requirement grows exponentially.

A. Related work

Transfer learning (TL)[3] shows the idea that to share knowledge among different tasks. In TL framework, learned knowledge in former tasks can be transferred though tasks of different settings, or even cross different types of tasks. By TL, RL learning agents are able to achieve faster learning performance. However, current transfer learning requires a lot of manual interventions, especially the mapping function among different tasks. Relational reinforcement learning (RRL)[4] will be a sound solution in this problem.

Decision tree technique is applied to reinforcement learning(DTRL) [12], [13] to split large amount of data from sensor reading into different states. At the leaf nodes of the tree, corresponding state values are stored. Therefore,decision tree technique saves memory for state representation. For this character, it is suitable for those tasks in which it is not easy to distinguish one state from others or those tasks that has so many states that looking up table will require very large memory. However, decision tree stores value for each individual task. It is not suitable for reusing in different tasks.

In addition, it is worth to mention fuzzy decision tree [9] which can classify data into different fuzzy sets, instead of crisp category in common decision tree.

B. Paper motivation

The idea of rule abstraction is not new. Fuzzy reinforcement learning(FRL)[6], [8], [7] has shown great advantage in solving navigation problems with fuzzy rules. But in current FRL research, the rule bases are either provided by human or enumerated across all possible combination of fuzzy terms. Therefore current fuzzy-RL can not be extended in different tasks.

The motivation of this work is similar with fuzzy reinforcement learning that to use rules to represent knowledge learned. However, different with FRL where rules are in the form that

If state is A then action is B
with corresponding value function v

we use the form that

If state is A and action is B
then its a Good Choice

It will be shown later that this form is convenient when decision tree is used to generate the rule base autonomously. We generate rules from agents' experienced state-action pairs. To archive autonomous rule generation, decision tree technique is applied.

We organize this paper as follow: Section II and III summary necessary techniques. Section describe problem configuration. Section present purposed method. Section presents simulation result and conclusion is discussed in Section.

II. REINFORCEMENT LEARNING

Reinforcement Learning framework[1] describes learning tasks which require series of action choice by Markov-Decision-Process. It includes:

1) A set of states S. It is the set of states that an agent would be in. There are initial states $\exists s \in S_{init}, S_{init} \subset S$ in which an agent starts and terminal

This work was partly supported by Global Center of Excellence for Mechanical Systems Innovation

Min Wu is with the Department of Precision Engineering, the University of Tokyo, Hongo 7-3-1, Tokyo, Japan wumin@robot.t.u-tokyo.ac.jp

Atsushi Yamasita is with Faculty of Precision Engineering, the University of Tokyo, Hongo 7-3-1, Tokyo, Japan yamasita@robot.t.u-tokyo.ac.jp

Hajime Asama is of Faculty of Departement of Precision Engineering, the University of Tokyo, Hongo 7-3-1, Tokyo, Japan asama@robot.t.u-tokyo.ac.jp

states $\exists s \in S_{term}, S_{term} \subset S$ in which an agent terminates learning.

2) A set of actions A. It is the set of actions an agent could execute.
3) A transfer function T. $T : S \times A \times S \mapsto R$ indicates the probability of the next state s_{t+1}, by taking action a_t, in state s_t. It is also called the model of environment.
4) Reward function R: $R : S \times A \mapsto R$ shows how an agent benefits from the environment by arriving in a certain state s by taking action a.
5) Discounted accumulated reward R_t: The accumulated reward is discounted from time t until the end of one learning process.

$$R_t = \sum_{i=t+1}^{n} \gamma^{i-t} r(i) \tag{1}$$

where $0 < \gamma \leq 1$ is discount factor, $r(i)$ is immediate reward defined by R.

6) A policy π: $\pi : S \mapsto A$ suggests an agent to take action a_t in state s_t by probability $\pi(s_t, a_t)$. For an agent, the purpose of reinforcement learning is to obtain optimal policy π^* which maximizes discounted accumulated reward R_t.
7) Value function $V(s)$ and $Q(s, a)$: Value function estimates average discounted reward under policy π for each state or state-action pairs. It evaluates how the agent can benefit from certain state s by $V(s) : S \mapsto R$ or from certain state s and taking action a by $Q(s, a) : S \times A \mapsto R$.

$$\begin{aligned} Q_\pi(s_t, a_t) &= E_\pi(R_t | s_t, a_t) \\ V_\pi(s_t) &= E_\pi(R_t | s_t) \end{aligned} \tag{2}$$

8) Episode: The full progress that an agent starts from $s_0 \in S_{init}$, by executing policy π and arrives in $s_n \in Sterm$.
9) Markov Property: The environment model should satisfy that:

$$\begin{aligned} &P\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \cdots, \\ &s_0, a_0, r_0\} = P\{s_{t+1} = s', rt + 1 = r | s_t, a_t, r_t\} \end{aligned} \tag{3}$$

10) Action Selection Policy: A well known policy is *Softmax* policy that:

$$p(s = s_t, a = a_i) = \frac{e^{-Q(s_t, a_i)}}{\sum_{j=1}^{n} e^{-Q(s_t, a_j)}}, \quad i = 1, \ldots, n \tag{4}$$

where $p(s = s_t, a = a_i)$ is the probability of selecting action a_i given state s_t.

By value function, RL problems of finding optimal policy π^* are converted to finding optimal value function V^*s or $Q^*(s, a)$ if environment satisfies equation (3) There are two kinds of value function estimation methods. The on-policy method which evaluate $V(s)$ and the off-policy method which evaluate $Q(s, a)$. Q-learning [7] is a well known off-policy algorithm that to estimate value function Qs_t, a_t in state s_t by taking action a_t at time t by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \tag{5}$$

where s_{t+1} is the state result by action a_t and r is the temporal reward by state transition $s_t \rightarrow s_{t+1}$

III. Decision Tree Technology

Decision tree is a supervised learning method that classifies a group of data Q which are represented by a set of features, or *attributes* A, into class $c_i \in C$. For each attribute $A_i \in A$, it has its value in a set V_i. Decision tree organises its classification progress by a series of testing if A_i is $v_i, v_i \in V_i$ in each node of tree, and classes c_i for each data in the leaf of each branch as well. Decision tree requires a set of training data to build. After a tree is build, further data can be automatically classified to certain class.

ID3[5] is an important learning algorithm in decision tree building. It deals with data of discrete domains that the value of each attribute A_i is discrete, and binary decision that class $c \in \{0, 1\}$. In each node, the testing attribute A_i of is chosen in the set of attributes that maximize information gain G. The information gain is the difference of information I between the data before and after classified by A_i. A summary of ID3 algorithm is presented below.

1) Given training data Q which is described by a set of attributes A, compute its information I by:

$$I_Q = -\frac{p}{N} log_2 \frac{p}{N} - \frac{n}{N} log_2 \frac{n}{N} \tag{6}$$

where $N = p + n$ is the number of data in set Q and there are p data are classified to 1 as well as n data are classified to 0.

2) Choose an attribute A_i in set A which has $|A_i|$ different values $V = \{v_1, \cdots, v_{|A_i|}\}$ to split the data Q into $|A_i|$ subsets $Q_j, j = 1, \cdots, |A_i|$ in which there is $A_i = v_j \in V$. In each subset Q_j there are N_j data with p_j classified to 1 and n_j classified to 0.
3) Compute information for each subset Q_j by:

$$I_{Q_j} = -\frac{p_j}{N_j} log_2 \frac{p_j}{N_j} - \frac{n_j}{N_j} log_2 \frac{n_j}{N_j}, j = 1, \cdots, |A_i| \tag{7}$$

4) Compute weighted summary information for all Q_j as the information I_i contained by data Q after split by attribute A_i by:

$$I_i = \sum_{j=1}^{|A_i|} \frac{N_j}{N} I_{Q_j} \tag{8}$$

5) *Information Gain* $G(A_i)$ is defined by :

$$Gain(A_i) = I_Q - I_i \tag{9}$$

6) Select the attribute A_k that maximize information gain(9) as the node of tree so that Q is divided into $|A_k|$ subsets $Q_j, j = 1, \cdots, |A_k|$.
7) For each subsets Q_j, repeat from setup 1) in attribute set $A - A_k$ unless all data in subset Q_j belong to the same class.

8) A decision tree can generate rules from its root to leafs by its testing condition in the form that:
if A_{root} is v_{root} and ... A_{leaf} is v_{leaf} then Class is c

IV. Proposed Method

A. Tree Generation

It is supposed that state of a learning agent can be expressed by n variables, or attributes, labelled by $V_1, V_2, \ldots, V_n$. That is, for each state $s \in S$, there is:

$$s = [v_1, v_2, \cdots, v_n]^T, \quad v_i \in V_i, i = 1, \ldots, n \tag{10}$$

In each state, the learning agent can choose an action a_i from m possible actions $a_1, \cdots, a_m$. Therefore by Q-learning algorithm (5), value functions of each state-action pair form a look-up table that:

V_1	V_2	$\cdots$	V_n	$Action$	$Q-Value$
v_{11}	v_{12}	$\cdots$	v_{1n}	a_1	Q_{s_1,a_1}
v_{11}	v_{12}	$\cdots$	v_{1n}	a_2	Q_{s_1,a_2}
		$\cdots$		$\cdots$	
		$\cdots$	v_{1n}	a_m	Q_{s_1,a_m}
v_{21}	v_{22}	$\cdots$	v_{2n}	a_1	Q_{s_2,a_1}
v_{21}	v_{22}	$\cdots$	v_{2n}	a_2	Q_{s_2,a_2}
		$\cdots$		$\cdots$	
		$\cdots$	v_{2n}	a_m	Q_{s_2,a_m}
		$\cdots$		$\cdots$	
		$\cdots$		$\cdots$	
		$\cdots$		$\cdots$	

For each state s (suffix for state index is dropped for convenience) that:

v_1	v_2	$\cdots$	v_n	a_1	Q_{s,a_1}	$Class_1$
v_1	v_2	$\cdots$	v_n	a_2	Q_{s,a_2}	$Class_2$
		$\cdots$		$\cdots$		
		$\cdots$	v_n	a_m	Q_{s,a_m}	$Class_m$

value function Q_{s,a_m} is categorized into two fuzzy sets labelled as *Good* and *Bad*. We define *CLASS* as the member ship of fuzzy set *Good*. For simplification, we assume that *Good* and *Bad* is a triangular orthogonal fuzzy set on *CLASS*. That is, when $CLASS = 1$ it means *Good* and when $CLASS = 0$ it means *Bad*. The *CLASS* is computed that, with state s and its possible action set $\{a_1, \cdots, a_m\}$:

$$\begin{aligned} &CLASS_i = 1 - \frac{Q_{max} - Q(s,a_i)}{Q_{max}}, i \in 1, \cdots, m \\ &where \quad Q_{max} = max(Q(s,a_1), \cdots, Q(s,a_m)) \end{aligned} \tag{11}$$

As defined in Equation (4), the class is computed within *one* sate with all its possible actions. This definition emphasizes the relativity character of fuzzy logic that a *Good* action may not necessarily to have a large value compared with all possible states.

Further more, some alternate of equation (4)may be introduced that we should eliminate those states that either action is *Bad*.Therefore, *CLASS* can be defined that,

$$if \quad \frac{max(Q(s,a_1), \cdots, Q(s,a_m))}{minQ_{all}} > h \quad then \quad CLASS = 0 \tag{12}$$

where $minQ_{all}$is the minimum value of all known state and h is a factor decided by human to express the degree of *Bad*.

To handle the situation that during exploring state of RL, some Q-function are of their initial value, such as 0, we follow the idea in fuzzy decision tree[9] that to set its class averagely that:

if $Q(s, a_i)$ is *unknown* then *CLASS*=0.5

B. Rule Generation

After categorizing value function table, we can use decision tree algorithm described in section III to build a tree to classify data. As in fuzzy decision tree, p and n in (6), which express number of data classified in class *Good* and in class *Bad* respectively, are computed by:

$$\begin{aligned} p &= \sum_i^N CLASS_i \\ n &= \sum_i^N (1 - CLASS_i) \end{aligned} \tag{13}$$

where N is the total umber of data.

We introduce two factor F_{pos} and F_{neg} to express the confidence of data belong to set *Good* and *Bad*. They are decided by human. Decision tree construction terminal condition is alternated to either all attributes are used or $p/N > F_{pos}$ or $p/N < F_{neg}$. When a decision tree is constructed, we can generalize rule from its root to leaves. The rules are in the form that:

if $V_1 = v_1$ and $V_2 = v_2$ and $\cdots$ and $V_t = v_t$ and $Action = a_i$ then CLASS is x*Good*

where $V_i, i = 1, \ldots, t$ are attributes of states, $a_i \in A$ is action and x is confidence of set *Good*. Furthermore, in the initial stage of RL, the Q-value table is so sparse that most of data are categorized into 0.5. Therefore rules in which only $x > F_{pos}$ or $x < F_{neg}$ is considered.

This operation results in two series of rules leading to *Good* choice and *Bad* choice. We label them as *Rule Good* and *Rule Bad*. When a learning agent decides its action, it is likely to try to chose actions in *Rule Good* and to avoid chosing actions in *Rule Bad*. That is, for probability $p(t) = 1 - e^{-f(t)}$ where t is passing time:

$$\begin{aligned} &P(s = s_t, a = a_i) = \frac{e^{Q'(s_t,a_t)/\tau}}{\sum_{b=1}^m Q'(s_t,a_t)/\tau} \\ &Q'(s_t, a_t) = N_{RuleGood}(s_t, a_i) - N_{RuleBad}(s_t, a_i) \end{aligned} \tag{14}$$

where s_t is current state, a_i is possible action. $N_{RuleGood}$ is the number of rules that suggested action a_i given s_t in set *Good*. $N_{RuleBad}$ is the number of rules that suggested action a_i given s_t in set *Bad*, respectively. This softmax approach will balance conflict rules in rule sets. The algorithm is shown as blow. An important difference between rule table and Q-value table is that in Q-value table, the number of attributes $|V_i|$ equals to the total number of attributes n. But in rule table, the number of attributes in each rule can be less than n. This character of rule table shows that by looking up

Algorithm 1 Algorithm Summary

repeat
  initial all $Q(s,a)$ STATE $s \leftarrow s = 0$
  repeat
    for each episode
    obtain s
    for probability $p = 1 - e^{-f(t)}$
    if $rand(1) < p$ **then**
      select action a by softmax (4)
    else
      obtain $N_{Good}(s,a)$ and $N_{Bad}(s,a)$ for each $a \in A$
      select a by (14)
    end if
    execute a at s, obtain r and new state s'
    $Q(s,a) \leftarrow Q - Learning(Q(s,a), s, a, s')$ by (5)
    record$s, a, Q(s,a)$ in DATA
    $s \leftarrow s'$
  until $s \in_{termial}$
  for all $< s, a, Q(s,a) >$ **do**
    obtain $class_{s,a,Qs,a}$ by (11)
  end for
  repeat
    obtain p, n by (13) and I by (6)
    for all attribute V_i **do**
      use V_i split DATA
      obtain p_i, n_i by (13) and I_i by (6)
    end for
    record V_i as a node of TREE
    use V_i maximize $I - I_i$ to split DATA
  until tree construction end
  generate *RuleGood* and *RuleBad* from TREE
until max learning loop reached

a Q-value table, a learning agent can only get information about one state-action pair but by looking up a rule table it can get information about all state-action pairs which satisfy the rule. Recall that the size of state space of RL is defined by:

$$size = |A| \prod_{i=1}^{n} |V_i| \tag{15}$$

therefore reducing number of attributes appears in look-up table will reduce the size of space to be searched into largely.

V. Robot Definition

Proposed task environment is shown in Fig. 1(a). A mobile learning agent(red circle) A is suggested to move in the ground defined by outer square to its goal point(green square) without collision with other obstacles. Obstacles are defined by circles with different radius in the ground. The agent is suggested to be equipped by a Laser Range Finder(LRF) to acquire distance information from obstacles and boundaries. It is also suggested that the agent knows where its goal is. Limited by LRF sensor function, the agent can only get distance information in the range of $240°$, with $120°$ on the left and $120°$ on the right, as it is shown in Fig. 1(b). Sensor range of LRF is divided into 5 zones called *Right*, *R-Front*, *Front*, *L-Front* and *Left* symmetrically. In each zone, the minimum distance reading d_i fro LRF is used to presentate distance information in the zone. d_i is discretized into 3 part, *Large*, *Middle* and *Small*.

Position information from robot to its goal poitn is defined in Fig. 1(c). The direction of robot motion is defined as *front*. *Right*, *Back* and *Left* are defined respectively. Distance from robot to its destination, d_{goal} is discretized by *Large*, *Middle* and *Small*, too.

Therefore the state of the robot can be described by:

$$d_R \quad d_{R-F} \quad d_F \quad d_{L-F} \quad d_L \quad d_{goal} \quad \theta_{goal}$$

where discreted distances ds can be any value in $\{Large, Middle, Small\}$ and θ_{goal} can be any value in $Front, Right, Back, Left$

The action space of the agent is $\{v, +\omega, -\omega\}$ indicating that the agent can move towards its direction at velocity v, or turn left, indicated by '+' or right, indicated by '-', at angle velicity ω. For simplicity it is supposed that the agent can not change its direction while moving.

In order to handle continuous time simulation in discrete state-space, technical of semi-MDP[10] is applied. That is, the agent will continuous execute its action until state change is detected. Accumulate reward is accounted by factor $\int \gamma^t dt$.

Reward structure of robot is decided as below. For every step the agent received -1 punishment from environment. When it research its goal, the reward is 0. If the agent collide with obstacles or boundaries, it receive -100 punishment. Therefore the reward is usually less than 0. Furthermore, $maxLearningTime$ is defined so that when the agent spends too much time in one episode, it stops and is marked as failure. When $maxLearningTime$ is reached, the agent gains -100 punishment.

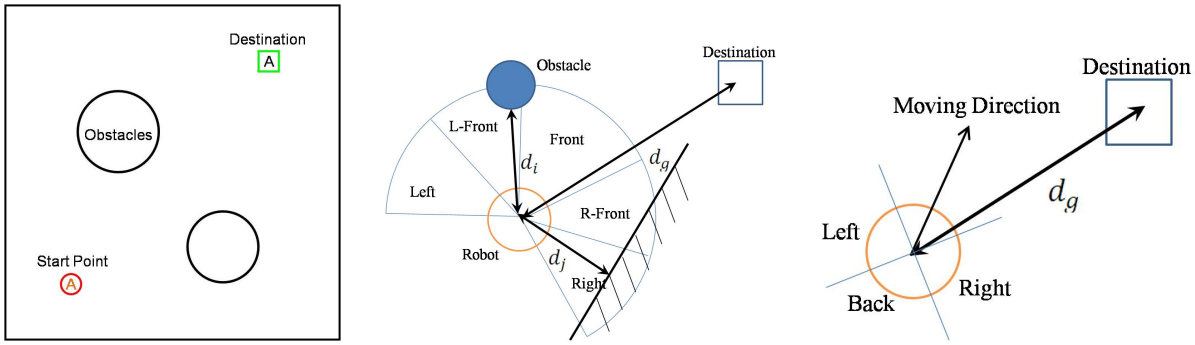
The setting of agent in simulation is listed below:

size	$r = 0.2m$
speed	$v = 0.5m/s$
	$\omega = 0.52rad/s$
Learning rate	$\alpha = 0.7$
Discount rate	$\gamma = 1$
Rule Factor	$F_{pos} = 0.8, F_{neg} = 0.2$
h	0.8

VI. Simulation Result

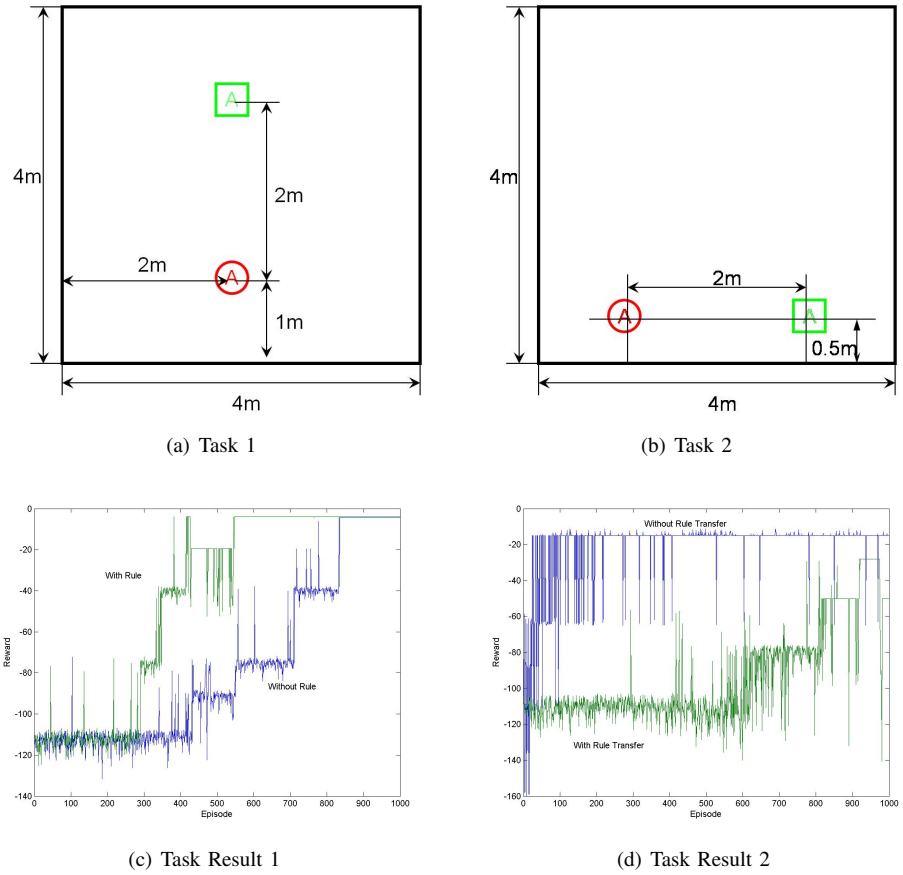
Two different environments are used to illustrated the character of RL with rule knowledge, as shown in Fig. 2(a) and Fig. 2(b). In the first task, an agent is placed just nearby its goal and there are no obstacles around it. The boundaries are initially far from the agent. In the second task, an agent is placed far away from its goal and very near to boundary on the right side. This task is similar to cliff walk task presented in [1]. While the first task is very easy, the second task is challenging to reinforcement learning because initial failure will result in large amount of search into useless state space.

Simulation setting is $t = 0.01s$, maximum trail time is $50s$ and in each task the agent repeat 1000 times. Simulation



(a) Environment setting (b) Distance from Objects (c) Goal setting

Fig. 1. Robot Configuration



(a) Task 1 (b) Task 2

(c) Task Result 1 (d) Task Result 2

Fig. 2. Task setting and result

environment is in C++. Each task runs more than 5 times and averaged data are collected.The result is drawn as x axis the number of trials and y axis is its reward. Fig. 2(c) shows result of task 1 shows that the for the simplest task, RL and purposed method works both well. The proposed method shows a bit better performance in learning rate compared by RL. However, in task 2, the average reward of purposed method is worse than pure RL learning algorithm. Figure 2(d) shows the proposed method as a bit higher initial performance but get low average reward in the later. It is because the proposed method is likely to force the agent to choose a very ”dangerous” path that is to move along the walls. However, as the agent is still exploring the environment and most of its state-action value is unknown therefore is labelled as 0.5*Good* therefore the agent is try to choose them which will have great probability to lead to collision with the wall on its right side. However, if we transfer the rule that the agent learned in task 1 to task 2, the situation changed. That is, in the very beginning of task 2, the agent's rule base is initialized by rules that it learned in task 1. Figure VI shows the result that with transfer, the agent achieved to learn faster and better final performance.

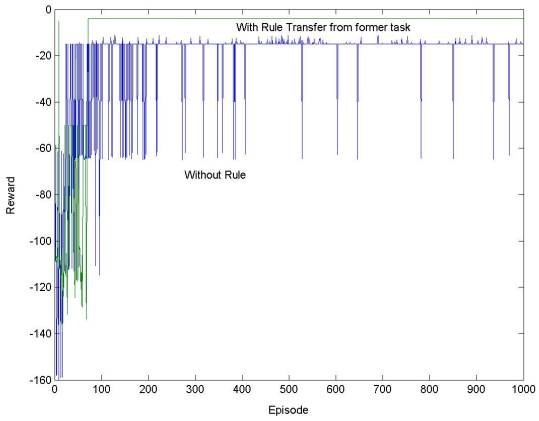


Fig. 3. Result of Task 2 by Transfer

VII. Conclusion and Discussion

In this paper, we present a method to construct rule base from reinforcement learning process as knowledge based on Q-learning and fuzzy decision tree techniques. It enables an agent to abstract rules from its experienced task, and reuse these rules in new tasks autonomously. The simulation result in section VI has shown that the purposed method enable the reinforcement learning agent to archive better performance.

However, there are some problem to be discussed. First of all, the result curve in either fig. 2(c), fig. 2(d) or VI shows burrs which infers that the agent come across some failures even if it has learned converging policy. It may be caused by that the reinforcement learning algorithm does not converge properly. As a reinforcement learning agent requires its state space to satisfy Markov property, laser range finder reading may cause non-Markov problem due to its partially observable character. Therefore, to achieve better result, POMDP framework [11] should be included into purposed method.

The second issue is that in transfer learning, mapping between different states are usually based on similarity between them. However, for autonomous transfer, the agent must be able to find which states can be mapped with each other. In our approach, an underline suggestion is that in new task, the learning agent regards any state in new tasks, which can satisfy all tests in one brunch of decision tree, is similar with those states of old tasks that satisfy the same tests in the same brunch of the tree. Therefore knowledge can be transferred. If old knowledge is not suitable for new environment, decision tree generating algorithm will correct the old tree to match new state-space.

The third issue on reinforcement learning is that in simulation, we find that initial value of each unknown state, as well as the punishment agent received when it come across the state *time over*, affect the convergence greatly. The learning algorithm converge much slower when value of all unknown states are initialized as 0. The learning algorithm is likely to repeat ending in *time over* if the punishment of *time over* is set smaller than the punishment of *fail*.

In summary, the purposed algorithm performs well in situations that reinforcement learning converges correctly. But if the value function can not be required correctly, the rule base may be ill and force the learning agent to make more failures. In order to overcome this problem, technique that help converge in ill environment, such as POMDP environment, should be introduced. Furthermore, human experience as well as social rules can also be regarded as initial rule base. These initial rules is expected to enable the learning agent to interact better in social environment.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[2] Kaelbling, Leslie Pack and Littman, Michael L and Moore, Andrew W, Reinforcement Learning : A Survey, Machine Learning, vol. 4, 1996, pp 237-285.

[3] Taylor, Matthew E and Stone, Peter, Transfer Learning for Reinforcement Learning Domains : A Survey, Journal of Machine Learning Research, vol. 10, 2009, pp 1633-1685.

[4] Džeroski, S. and De Raedt, L. and Driessens, K., Relational reinforcement learning, Machine Learning, vol 43, no. 1, 2001, pp 7-52.

[5] J.R. Quinlan, Induction of Decision Trees, Machine Learning, vol. 1, no. 1, 1986, pp 81-106

[6] Beom, H. Rak and Cho, H. S., A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning, IEEE Trans. Syst., Man, Cybern., vol. 25, no. 3, 1995, pp 464-477.

[7] Berenji, H.R. and Khedkar, Pratap, Learning and tuning fuzzy logic controllers through reinforcements, IEEE Trans. Neural Netw., vol. 3, no. 5, 2002, pp 724-740.

[8] Faria, G and Romero, RAF, Incorporating fuzzy logic to reinforcement learning, Fuzzy Systems 2000. the 9th IEEE International Conference on, vol. 2, 2000, pp 847-852.

[9] C. Z. Janikow, Fuzzy decision trees: issues and methods, IEEE Trans. Syst., Man, Cybern. B, vol. 28, no. 1, 1998, pp 1-14.

[10] S.J. Bradtke, M.O. Duff, Reinforcement learning methods for continuous-time Markov decision problems,in: NIPS-7, MIT Press, Cambridge, MA, 1995, pp. 393400

[11] Cassandra, A. R. A Survey of POMDP Applications. Operations Research. 1998. pp. 1724

[12] McCallum, R. Overcoming incomplete perception with utile distinction memory. Proceedings of the Tenth International Conference on. 1993

[13] Pyeatt, L. D., and Howe, A. E. Decision Tree Function Approximation in Reinforcement Learning. Proceedings of the Third International Symposium on Adaptive Systems Evolutionary Computation and Probabilistic Graphical Models 2001 Vol. 2, pp. 7077