Simulation-Based Evaluations of Reinforcement Learning Algorithms for Autonomous Mobile Robot Path Planning

Hoang Huu Viet, Phyo Htet Kyaw and TaeChoong Chung

Abstract This work aims to evaluate the efficiency of the five fundamental reinforcement learning algorithms including Q-learning, Sarsa, Watkins's Q(λ), Sarsa(λ), and Dyna-Q, and indicate which one is the most efficient of the five algorithms for the path planning problem of autonomous mobile robots. In the sense of the reinforcement learning algorithms, the Q-learning algorithm is the most popular and seems to be the most effective model-free algorithm for a learning robot. However, our experimental results show that the Dyna-Q algorithm, a method learns from the past model-learning and direct reinforcement learning is particularly efficient for this problem in a large environment of states.

Keywords Reinforcement learning · Autonomous mobile robots · Path planning

1 Introduction

Mobile robotics is a research area that deals with autonomous and semi-autonomous navigation. Path planning problem is recognized as one of the most fundamental problems to applications of autonomous mobile robots. The path

H. H. Viet (✉) · P. H. Kyaw · T. Chung
Artificial Intelligence Lab, Department of Computer Engineering,
School of Electronics and Information, Kyung Hee University,
1-Seocheon, Giheung, Yongin, Gyeonggi 446–701, South Korea
e-mail: viethh@khu.ac.kr

P. H. Kyaw
e-mail: phyo@khu.ac.kr

T. Chung
e-mail: tcchung@khu.ac.kr

planning or trajectory planning problem of autonomous mobile robots refers to determining a collision-free path from its position to a goal position through an obstacle environment without human intervention [1].

Reinforcement learning (RL) is an approach to artificial intelligence that emphasizes learning by an agent from its interaction with the environment [2, 3]. The goal of the agent is to learn what actions to select in situations by learning a value function of situations or “states”. The learning agent is not conducted which actions to take, but it has to discover an optimal action of each state which yields the high rewards in a long-term objective. In literature, there have been several RL algorithms suggested to solve the path planning problem of autonomous mobile robots. Among those algorithms of RL, the Q-learning algorithm [4] has been frequently employed to solve the path planning problem [5–8]. The strength of RL methods is that it does not require an explicit model of an environment, thus it can be popularly employed to solve the mobile robot navigation problem. However, one primary difficulty faced by RL applications is that the most RL algorithms learn very slowly. As such, this work aims to evaluate five popular algorithms of RL including Q-learning, Sarsa, Watkins’s Q(λ), Sarsa(λ), Dyna-Q based on computer simulations for the path planning problem of autonomous mobile robots, and to indicate which one is the most efficient for this problem. The rest of this article is organized as follows: Sect. 2 shows a short review of the algorithms that are going to be evaluated in this article. The evaluations are discussed in Sect. 3. Finally, we conclude our work in Sect. 4.

2 Background

2.1 Basic Concepts

Reinforcement learning emphasizes the learning process of an agent through trial-and-error interactions with an environment. In the standard RL model, an agent connects to its environment via perceptions and actions. On each step of interaction the agent receives a *state*, *s*, of the environment as an input and then the agent takes an *action*, *a*. The action changes the state of the environment, and a scalar value of the state-action pair is sent to the agent, called a *reward function r* of the state-action (*s, a*) pair. The set of all states makes the *state space, S,* of the environment, and the set of actions of the state *s* makes the *action space*, *A(s)*. The *value function* of a state (or state-action pair) is the total amount of rewards that an agent can expect to accumulate over the future starting from that state. A reward function indicates what is good in an immediate sense, whereas a value function specifies what is good in the long-run. A *policy* is a mapping from perceived states of the environment to actions taken in those states. A *model* of the environment is something that mimics the behavior of the environment. Given a state and an action, the model might predict the resultant of the next state and the reward function. The objective of the agent is to learn actions that tend to maximize the long-run sum of the value of the rewards.

An *on-line* learning method learns while gaining experience from the environment. An *off-line* learning method waits until it is finished gaining experience to learn. An *on-policy* learning method learns about the policy it is currently following. An *off-policy* learning method learns about a policy while following another.

A *greedy strategy* refers to a strategy that agent always chooses the action with the highest value of the value function. The selected action refers to a *greedy action* and it is said that the agent is *exploiting* the environment. An ε-*greedy strategy* refers to a strategy that agent chooses the *greedy action* with probability of $1\text{-}\varepsilon$, and chooses the *random action* with a small probability of ε. The random action refers to a *non-greedy* action and it is said that the agent is *exploring* the environment.

If the agent-environment interaction process is broken into subsequences, each subsequence refers to an *episode* and the end state of each subsequence is called the *terminal state*. The learning task broken into episodes is called *episodic tasks*. In episodic tasks, the state space S denotes the set of all *non-terminal states* and the state space S^+ denotes the set S plus the *terminal state*.

In the RL algorithms, the parameter $\alpha \in (0, 1)$ denotes the learning rate, the parameter $\gamma \in (0, 1)$ denotes the discount rate, the parameter δ denotes the temporal-difference error, the parameter $\lambda \in (0, 1)$ denotes the decay-rate parameter for eligibility traces, the parameter ε denotes probability of random action in ε-*greedy* strategy, and $Q\ (s,\ a)$ denotes the action-value function of taking action a in state s.

2.2 Temporal Difference Learning Algorithms

In the temporal difference (TD) learning approach, two algorithms that can be identified as the main idea of TD method would certainly be Sarsa and Q-learning. The Sarsa algorithm (short for *state*, *action*, *reward*, *state*, *action*) is an *on-policy* TD learning algorithm, whereas the Q-learning algorithm is an *off-policy* TD learning algorithm. These two algorithms consider transitions from a state-action pair to a state-action pair and learn the action-value function of state-action pairs. While the Sarsa algorithm backups up the Q-value corresponding to the next selected action, the Q-learning algorithm backups up the Q-value corresponding to the action of the best next Q-value. Since these algorithms need to wait only one time step to backup the Q-value. So, they are *on-line* learning methods. The algorithms Sarsa [3] and Q-learning [4] are shown in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1: Sarsa algorithm

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
Initialize $s$
$a \leftarrow \varepsilon\text{-}greedy(s,Q)$
Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    $a' \leftarrow \varepsilon\text{-}greedy(s',Q)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
Until $s$ is terminal
```

Algorithm 2: Q-learning algorithm

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
Initialize $s$
Repeat (for each step of episode):
    $a \leftarrow \varepsilon\text{-}greedy(s,Q)$
    Take action $a$, observe $r$, $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
Until $s$ is terminal
```

2.3 Eligibility Traces

Algorithm 3: Sarsa(λ) algorithm

```
Initialize $Q(s,a)$ and $e(s,a) = 0$, for all $s$, $a$
Repeat (for each episode):
Initialize $s$, $a$
Repeat (for each step of episode):
Take action $a$, observe $r$, $s'$
 $a' \leftarrow \varepsilon\text{-}greedy(s',Q)$
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
$e(s,a) \leftarrow 1$
For all $s$, $a$:
    $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
    $e(s,a) \leftarrow \gamma\lambda e(s,a)$
$s \leftarrow s'$; $a \leftarrow a'$;
Until $s$ is terminal
```

Algorithm 4: Q(λ) algorithm

```
Initialize $Q(s,a)$ and $e(s,a) = 0$, for all $s$, $a$
Repeat (for each episode):
Initialize $s$, $a$
Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    $a' \leftarrow \varepsilon\text{-}greedy(s',Q)$
    $a^* \leftarrow \text{argmax}_b\, Q(s',b)$
    $\delta \leftarrow r + \gamma Q(s',a^*) - Q(s,a)$
$e(s,a) \leftarrow 1$
    For all $s$, $a$:
        $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
        If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
        Else $e(s,a) \leftarrow 0$
$s \leftarrow s'$; $a \leftarrow a'$;
Until $s$ is terminal
```

Eligibility traces are one of the basic mechanisms of RL. Almost any TD method can be combined with eligibility traces to obtain a more general method that may learn more efficiently. An eligibility trace is a temporary record storing a trace of the state-action pairs taken over time. When eligibility traces are augmented with the Sarsa algorithm, it is known as the Sarsa(λ) algorithm. The basic algorithm is similar to the Sarsa algorithm, except that backups which are carried out over n steps later instead of one step later. The Watkins's Q(λ) [hereinafter called Q(λ)] algorithm is similar to the Q-learning algorithm, except that it is supplemented eligibility traces. The eligibility traces are updated in two steps. First, if a *non-greedy* action is taken, they are set to zero for all state-action pairs. Otherwise, they are decayed by $\gamma\lambda$. Second, the eligibility trace corresponding to the current state-action pair is reset to 1. The algorithms Sarsa(λ) and Q(λ), referred from [3], using replacing traces are shown in Algorithm 3 and Algorithm 4, respectively.

2.4 Dyna-Q Algorithm

The Dyna-Q algorithm is the integration of planning and direct RL methods. Planning is the process that takes a model as an input and produces a policy by using simulated experience generated uniformly at random, whereas direct RL method uses a real experience generated by the environment to improve the value function and policy. The Dyna-Q algorithm is shown in Algorithm 5 [3]. The *Model(s, a)* represents the next predicted state and reward of the model for the state-action pair *(s, a)* and N is the number of planning steps. Step (d) is the direct RL, steps (e) and (f) are model-learning and planning, respectively. If steps (e) and (f) are omitted, the planning step $N = 0$, the remaining algorithm is the Q-learning algorithm.

Algorithm 5: Dyna-Q algorithm

```
Initialize $Q(s,a)$ and $Model(s,a)$, for all $s$, $a$
Do forever:
(a) $s \leftarrow$ current (non-terminal) state
(b) $a \leftarrow \varepsilon\text{-}greedy(s,Q)$
(c) Take action $a$, observe $r$, $s'$
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat $N$ times:
    $s \leftarrow$ random previously observed state
    $a \leftarrow$ random action previously taken in $s$
    $s', r \leftarrow Model(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
```

3 Evaluations

In this section, assumptions of the path planning problem are defined. Evaluations based on simulations of the algorithms are implemented to determine which one is the most efficient for the autonomous mobile robot path planning.

3.1 Assumptions

Assumption 1 The environment of the robot consists of a goal position and obstacles. The position of the goal, the position and shape of obstacles are unknown by the robot.

Assumption 2 The robot is equipped with all necessary sensors such that the robot knows its position, detects obstacles if collisions occur, and determines the goal if it reaches to the goal position during navigating time.

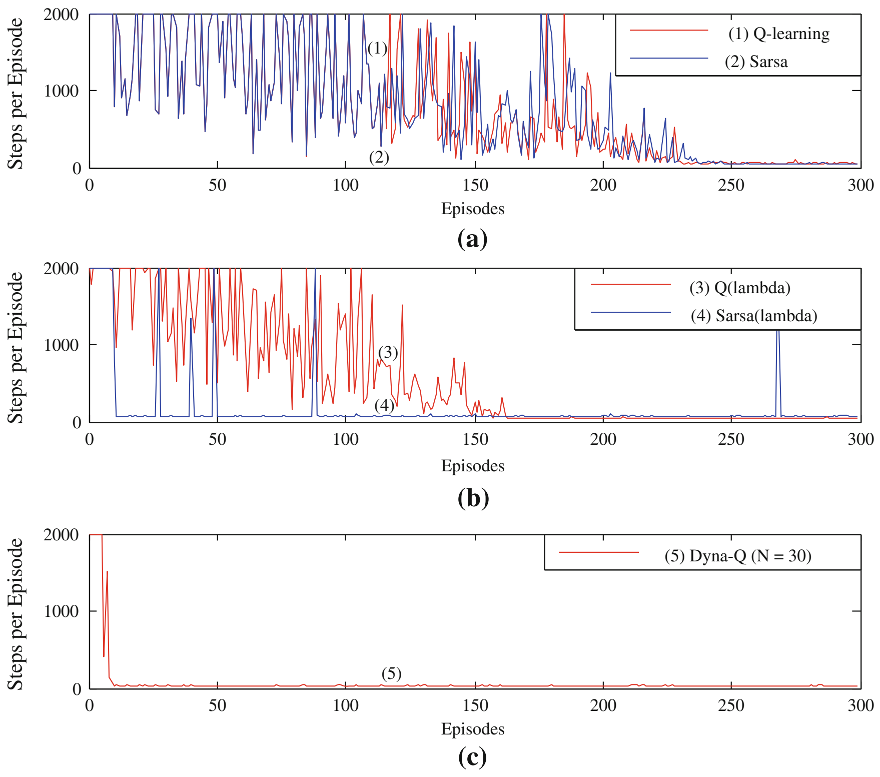

Fig. 1 Comparison steps per episodes of algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q

Table 1 The performance of the algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), Dyna-Q described in the first simulation

Criterion	Q-learning	Sarsa	Q(λ)	Sarsa(λ)	Dyna-Q
Episodes	230	250	170	270	10
Steps	227, 469	237, 401	170, 234	48, 653	14, 138
Path length	53	49	41	62	34

Assumption 3 The robot initially has no knowledge of the effect of its actions on what position it will occupy next and the environment provides rewards to the robot and that this reward structure is also initially unknown to the robot.

Assumption 4 From its current position, the robot can move to an adjacent position in one of the eight directions, East, North-East, North, North-West, West, South-West, South, and South-East, except that any direction that takes the robot into obstacles or outside of environment, in which case the robot keeps its current position.

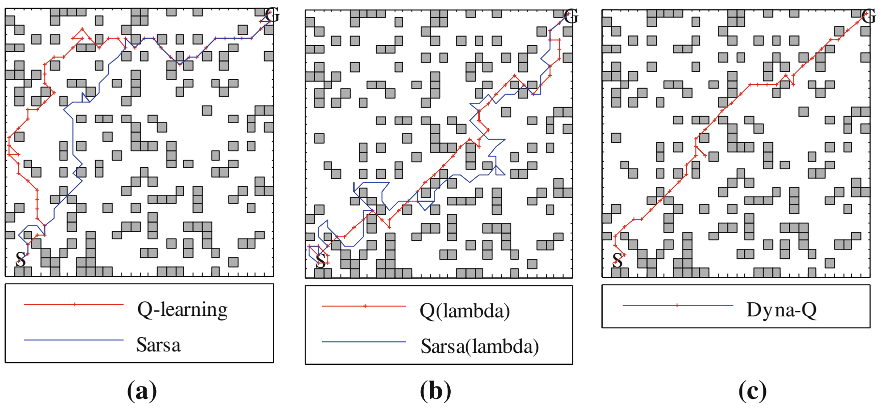

Fig. 2 Paths are found by the algorithms **a** Q-learning and Sarsa, **b** Q(λ) and Sarsa(λ), **c** Dyna-Q after 300 episodes

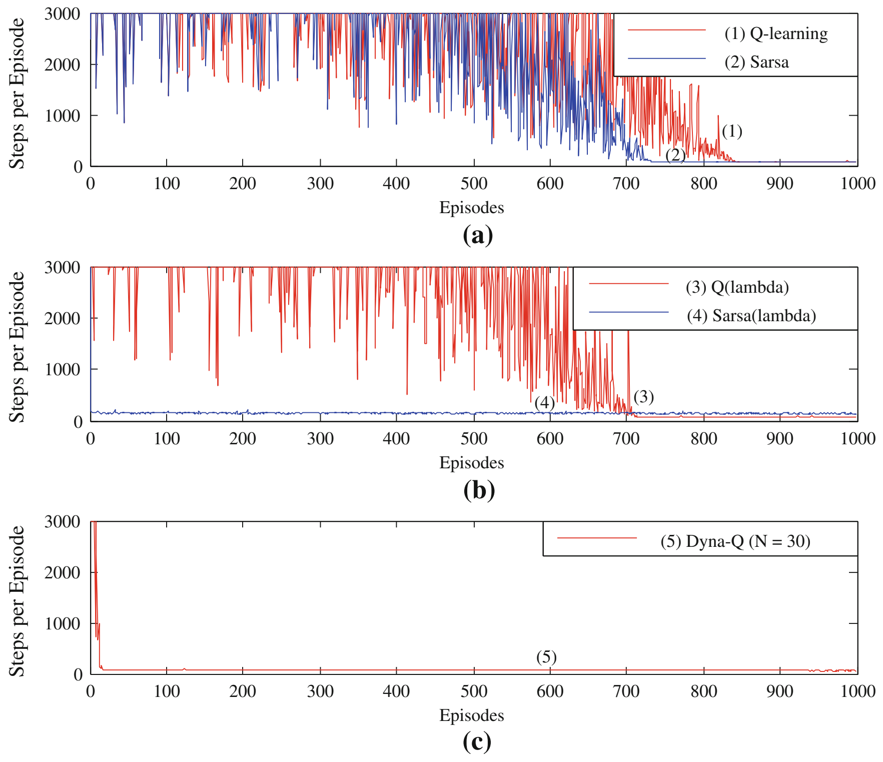

Fig. 3 Comparison steps per episodes of algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q

Table 2 The performance of the algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), Dyna-Q described in the second simulation

Criterion	Q-learning	Sarsa	Q(λ)	Sarsa(λ)	Dyna-Q
Episodes	850	720	710	10	20
Steps	1, 963, 980	1, 731, 725	1, 728, 585	6, 757	30, 556
Path length	74	73	74	128	58

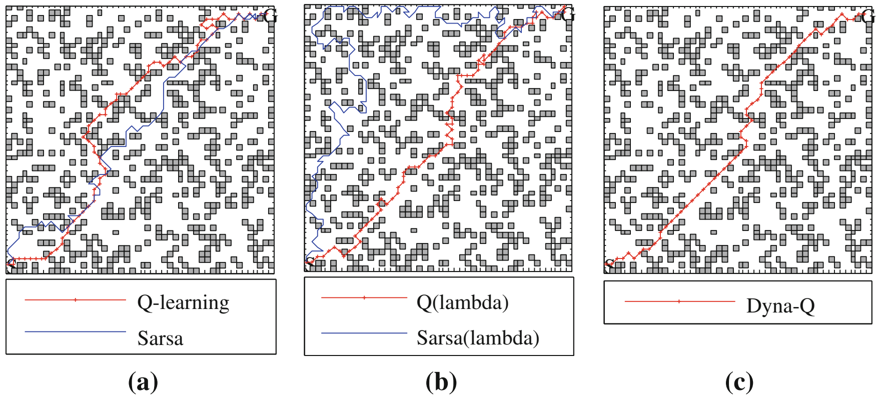

Fig. 4 Paths are found by the algorithms **a** Q-learning and Sarsa, **b** Q(λ) and Sarsa(λ), **c** Dyna-Q after 1,000 episodes

Assumption 5 If the robot reaches to the goal position, a reward of 1 is given for the robot. Otherwise, a reward of zero is given for it. After reaching the goal position, the robot returns to the start position to begin a new episode.

The task of the robot is to discover a collision-free path from the start position (S) to the goal position (G) through its environment. Evaluations of algorithms for the path planning problem are based on the speed of convergence of the algorithms to a near-optimality path and length of the path obtained.

3.2 *Simulations and Evaluations*

In this section, two simulations using the Matlab software are implemented to evaluate the efficiency of the algorithms. The environments of these simulations are represented by the cells of a uniform grid. Each cell with a zero value is considered as a state of the environment. Otherwise, it is considered as an obstacle. The basic parameters for the all simulations are set as follows: $\alpha = 0.1$, $\gamma = 0.95$, $\lambda = 0.95$, $\varepsilon = 0.05$. After each episode, the value of ε is set again by $\varepsilon = 0.99\varepsilon$.

The environment of the first simulation is a maze as shown in Fig. 2. The maze consists of $30 \times 30 = 900$ cells in which 20% cells make obstacles, so the number of states of the environment is 720 states. The maximum step of each episode is

Table 3 The evaluations of the algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q

Criterion	Q-learning	Sarsa	Q(λ)	Sarsa(λ)	Dyna-Q
Soundness	Yes	Yes	Yes	Yes	Yes
Completeness	Yes	Yes	Yes	Yes	Yes
Optimality	Bad	Bad	Medium	Bad	Good
Speed of convergence	Slow	Slow	Medium	Rapid	Rapid

2,000 steps. Figure 1 shows the steps per episodes of algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q. Table 1 depicts the performance of these algorithms, where episodes refer to the number of episodes taken to converge to a near-optimality path, steps refer to the sum of steps taken to converge to a near-optimality path, and path length refers to the length of the path found by the algorithms after 300 episodes. Figure 2 depicts the paths found by the algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q after 300 episodes. It can be seen from Table 1 and Fig. 2 that the Dyna-Q algorithm obtains a near-optimality path with the shortest length in the smallest number of steps among five algorithms.

The next simulation is to evaluate the efficiency of the five algorithms in a larger environment of states and obstacles. In this simulation, the environment is a maze as shown in Fig. 4. The maze consists of $50 \times 50 = 2{,}500$ cells in which 25% cells make obstacles, so the number of states of the environment is 1,875 states. The maximum step of each episode is 3,000 steps. Figure 3 shows the steps per episodes of algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q. Table 2 depicts the performance of these algorithms, where parameters are the same as in Table 1, except path length refers to paths found by the algorithms after 1,000 episodes. Figure 4 depicts the paths found by the algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q after 1,000 episodes. In this simulation, the Sarsa(λ) algorithm converges to a near-optimality path quickly, but the path found by this algorithm is much longer than the path found by the Dyna-Q algorithm. The Dyna-Q algorithm is really effective in this simulation.

Based on simulation results shown above, some evaluation criteria [1] of these algorithms are summarized in Table 3, where the soundness means that the planned path is guaranteed to be collision-free, the completeness means that the algorithm is guaranteed to find a collision-free path if one exists, the optimality means that the length of the actual path obtained versus the optimal path, and speed of convergence means that the computer time taken to find a near-optimality path. Here, the criteria of optimality and speed of convergence to a near-optimality path are only compared among the algorithms.

4 Conclusions

In this work, we reviewed and evaluated some popular RL algorithms for the path planning problem of autonomous mobile robots. In the first sense of RL algorithms, the Q-learning algorithm is the most popular and seems to be the most effective model-free algorithm for a learning robot. However the simulation results show that the Q-learning is not really effective for finding a collision-free path in an environment that the number of states and obstacles are so large. Both the Sarsa algorithm and the Q-learning algorithm converge quite slowly and the paths found by these two algorithms are not good paths. The algorithms Q(λ) and Sarsa(λ) improve quite well the speed of convergence to a near-optimality path. But, the Dyna-Q algorithm is particularly efficient in solving the path planning problem of autonomous mobile robots. With the experimental results shown above, we believe that the Dyna-Q algorithm is the best choice among algorithms Q-learning, Sarsa, Q(λ), Sarsa(λ), and Dyna-Q to solve the path planning problem. However, we have just emphasized our work on the simulations of the maze domain. We plan to extend the Dyna-Q algorithm to the real robot in the real environment.

Acknowledgments This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0012609).

References

1. Dudek G, Jenkin M (2010) Computational principles of mobile robotics. Cambridge University Press, New York
2. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
3. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. The MIT Press, Cambridge
4. Watkins C (1989) Learning from delayed rewards. Ph.D. Dissertation, King's College
5. Smart WD, Kaelbling LP (2002) Effective reinforcement learning for mobile robots. In: IEEE international conference on robotics and automation (ICRA'02), vol 4. IEEE Press, Washington, pp 3404–3410
6. Zamstein L, Arroyo A, Schwartz E, Keen S, Sutton B, Gandhi G (2006) Koolio: path planning using reinforcement learning on a real robot platform. In: 19th Florida conference on recent advances in robotics, Florida
7. Chakraborty IG, Das PK, Konar A, Janarthanan R (2010) Extended Q-learning algorithm for path-planning of a mobile robot. In: LNCS, vol 6457. Springer, Heidelberg, pp 379–383
8. Mohammad AKJ, Mohammad AR, Lara Q (2011) Reinforcement based mobile robot navigation in dynamic environment. Robotics Comput-Integr Manuf 27:135–149