# Extended Q-Learning Algorithm for Path-Planning of a Mobile Robot

Indrani Goswami (Chakraborty)[1], Pradipta Kumar Das[2],
Amit Konar[1], and R. Janarthanan[3]

[1] ETCE Department, Jadavpur University, Kolkata, India
[2] Dhaneswar Rath Institute of Engineering and Management Studies
Tangi, Cuttack, Orissa
[3] Jaya Engineering College, Tamilnadu
daspradipta78@gmail.com, konaramit@yahoo.co.in,
srmjana_73@yahoo.com

**Abstract.** In this paper, we study the path planning for Khepera II mobile robot in a grid map environment using an extended Q-learning algorithm. The extension offers an additional benefit of avoiding unnecessary computations involved to update the Q-table. A flag variable is used to keep track of the necessary updating in the entries of the Q-table. The validation of the algorithm is studied through real time execution on Khepera-II platform. An analysis reveals that there is a significant saving in time- and space- complexity of the proposed algorithm with respect to classical Q-learning.

**Keywords:** Q-learning, Reinforcement learning, Motion planning.

## 1 Introduction

Motion planning is one of the important tasks in intelligent control of a mobile robot. The problem of motion planning is often decomposed into path planning and trajectory planning. In path planning, we need to generate a collision-free path in an environment with obstacles and optimize it with respect to some criterion [8], [9]. However, the environment may be imprecise, vast, dynamical and partially non-structured [7]. In such environment, path planning depends on the sensory information of the environment, which might be associated with imprecision and uncertainty. Thus, to have a suitable path planning scheme in a cluttered environment, the controller of such kind of robots must have to be adaptive in nature. Trajectory planning, on the other hand, considers time and velocity of the robots, while planning its motion to a goal. It is important to note that path planning may ignore the information about time/velocity, and mainly considers path length as the optimality criterion. Several approaches have been proposed to address the problem of motion planning of a mobile robot. If the environment is a known static terrain and it generates a path in advance, it is said to be off-line algorithm. It is called on-line, if it is capable of producing a new path in response to environmental changes.

K. Deb et al. (Eds.): SEAL 2010, LNCS 6457, pp. 379–383, 2010.

A good path-planning algorithm requires *a priori* knowledge of the robot's world map. Usually a learning algorithm is employed to make the robot aware about its environment. Reinforcement learning [1-5] helps a robotic agent to adapt it experience throughout its life. Q-learning falls under the reinforcement class. This research is aimed at improving the performance of the Q-learning algorithm. It examines the scope of the improved algorithm in path planning application of mobile robots.

The rest of the paper is organized as follows. Classical Q-Learning is introduced in section 2. The algorithm for conditional Q-learning is presented in detail in section 3. Experimental instances are given in section 4, and conclusions are listed in section 5.

## 2 The Classical Q-Learning Algorithm

In classical Q-learning, all possible states of an agent and its possible action in a given state are deterministically known. In other words, for a given agent A, let $S_0$, $S_1$, $S_2$,..., $S_n$, be n- possible states, where each state has m possible actions $a_0$, $a_1$, $a_2$,...,$a_m$. At a particular state-action pair, the specific rewards that the agent may acquire is known as immediate reward. For example r ($S_i$, $a_j$) denotes the immediate reward that the agent A acquires by executing an action $a_j$ at state $S_i$. An agent selects its next state from its current states by using a policy. This policy attempts to maximize the cumulative reward that the agent could have in subsequent transition of states from its next state. For example, let the agent be in state Si and is expecting to select the next best state. Then the Q-value at state $S_i$ due to action of $a_j$ is given in equation 1.

$$Q(S_i, a_j) = r(S_i, a_j) + \gamma \underset{a'}{Max}\, Q(\delta(S_i, a_j), a') \tag{1}$$

where $\delta(S_i, a_j)$ denotes the next state due to selection of action $a_j$ at state $S_i$. Let the next state selected be $S_k$. Then $Q(\delta(S_i, a_j), a') = Q(S_k, a')$. Consequently selection of $a'$ that maximizing $Q(S_i, a_j)$ is an interesting problem. One main drawback for the above Q-learning is to know the Q-value at a state $S_k$ for all possible action $a'$. As a result, each time it accesses the memory to get Q-value for all possible actions at a particular state to determine the most appropriate next state. So it consumes more time to select the next state. Since the action $a'$ for which $Q(S_k, a')$ is maximum needs to be evaluated, we can remodel the Q-learning equation by identifying the $a'$ that moves the agent closer to the goal.

In the extended Q-learning to be presented, we have created only one field for action of each state. In this way, we save the space required for the Q-table. Thus the Q-table storing the Q-values for the best action in each state, requires small time for retrieval of Q-values, and hence saves significant time complexity.

## 3 The Extended Q-Learning

Let for any state $S_k$, the distance between the goal state and the next feasible state of $S_k$ are known. Let the next feasible state of $S_k$ be $\{S_a, S_b, S_c, S_d\}$. Let G be the goal and the distance between $S_a$, $S_b$, $S_c$, $S_d$ and G be $d_{aG}, d_{bG}, d_{cG}, d_{dG}$ respectively. Let the

distance in order be $d_{bG} < d_{aG} < d_{cG} < d_{dG}$ . Then the agent should select the next state $S_b$ from its current state $S_k$, if the Q-value of the state $S_b$ is known. We can evaluate the Q-value of state $S_k$ by the following approach.

$$\begin{aligned} Q(S_k, a^{/}) &= r(S_k, a^{/}) + \gamma \underset{a^{//}}{Max} Q(\delta(S_k, a^{/}), a^{//}) \\ &= 0 + \gamma \underset{a^{//}}{\mathrm{Max}}\, \mathrm{Q}(\delta(\mathrm{S_k}, a^{/}), a^{//}) \end{aligned} \tag{2}$$

Now $\delta(\mathrm{S_k}, a^{/}) = \mathrm{S}_a \mid \mathrm{S_b} \mid \mathrm{S_c} \mid \mathrm{S_d}$, where $\mid$ denotes OR operator. Therefore,

$$\begin{aligned} &\underset{a^{//}}{Max} Q(\delta(S_k, a^{/}), a^{//}) \\ &= \underset{a^{//}}{\mathrm{Max}}\, \mathrm{Q}\{\mathrm{S}_a \mid \mathrm{S_b} \mid \mathrm{S_c} \mid \mathrm{S_d}, a^{//}\} \\ &= \mathrm{Q}(\mathrm{S_b}, a^{//}). \quad (\because d_{bG} < \mathrm{d_{aG}} < d_{cG} < d_{dG}) \end{aligned} \tag{3}$$

Combining (2) and (3) we have :

$$\mathrm{Q}(\mathrm{S_k}, a^{/}) = 0 + \gamma\, \mathrm{Q}(\mathrm{S_b}, a^{//}).$$

Thus if the next state having the shortage distance with the goal is known, and the Q-value of this state is also known, then the Q-value of the current state is simply $\gamma \times$Q-value of this next state.

Let $S_p$, $S_n$ and $S_G$ be the present, next and the goal states respectively. Let $Q_p$ and $Q_n$ be the Q-value at the present and the next states $S_p$ and $S_n$ respectively. Let $d_{xy}$ be the Euclidean distance between the states $S_x$ and $S_y$. We use a Boolean variable Lock: $L_x$ to indicate that the $Q_x$ value of a state is fixed permanently. We set lock $L_n$=1 if the Q-value of the state n is fixed, and won't change further after $L_n$ is set to 1. The Lock variable for all states except the goal will be initialized zero in our proposed Q-learning algorithm.

We now study some interesting property of Q-learning algorithm, which would provide us an insight to extend classical Q-learning algorithm for both speed up and minimum space complexity. The proof of the properties is not given here for space limitation.

Property1 : *If* $L_n = 1$ *and* $d_{pG} > d_{nG}$ *then* $Q_p = \gamma \times Q_n$ and set $L_p = 1$.

Property2 : *If* $L_n = 0$ *and* $d_{pG} > d_{nG}$ *then* $Q_p = Max(Q_p, \gamma \times Q_n)$.

Property3 : *If* $L_p = 1$ *and* $d_{nG} > d_{pG}$ *then* $Q_n = \gamma \times Q_p$ *and set* $L_n = 1$.

Property 4 : *If* $L_p = 0$ *and* $d_{nG} > d_{pG}$ *then* $Q_n = Max(Q_n, \gamma \times Q_p)$.

It is apparent from the property 1 that if $L_n$ is 1, we on updating $Q_p$, set $L_p$ to 1. Similarly, in property 3, we set $L_n$ to 1 when $L_p$ =1 is detected. When $L_n$= 0 in property 2 and $L_p$ =0 in property 4, we just update $Q_p$ and $Q_n$ respectively. When $L_i$ for all state i=1, we need not update Q-table further. These properties have been employed in the extended Q-learning algorithm.

**Time-Complexity:** In classical Q-learning, the updating of Q-values in a given state requires determining the largest Q-value, in that cell for all possible actions. Thus if there are m possible actions at a given state, maximization of m possible Q-values, require m −1 comparison. Consequently, if we have n states, the updating of Q values of the entire Q table by classical method requires n(m −1) comparisons. Unlike the classical case, here we do not require any such comparison to evaluate the Q values at

a state $S_p$ from the next state $S_n$. But we need to know whether state n is locked i.e., Q-value of $S_n$ is permanent and stable. Thus if we have n number of states, we require n number of comparison. Consequently, we save $n(m-1) - n = nm - 2n = n(m-2)$.

**Proposed Algorithm for Extended Q-Learning**

1. Initialization
   For all $S_i, i = 1$ to n except $S_i = S_G$
   $\{\text{set } L_i = 0; Q_i = 0; \text{action} = \varphi\}$;
   $L_G$ (for goal $S_G$) $= 1; Q_G$(for goal) $= 100$;
2. Assign $\gamma$ in $(0,1)$ and initial state $= S_p$;
3. Update $Q-$table :
   Repeat
   { a)Select $a_i$ from $A = \{a_1, a_2, ..., a_m\}$;
   b)Determine $d_{nG}$ and $d_{pG}$;
   If $(d_{nG} < d_{pG})$
   Then if $L_n = 1$
   Then $\{Q_p = \gamma \times Q_n; L_p = 1;$ save action of $S_p = "a_i"\}$
   Else if $(Q_p > \gamma \times Q_n)$ then {save $Q_p = \gamma \times Q_n$ and action of $S_p = "a_i"\}$;
   Else if $L_p = 1$ then $\{Q_n = \gamma \times Q_p;$ save action of $S_n =$ opposite action of $"a_i"; L_n = 1\}$;
   Else if $(Q_n > \gamma \times Q_p)$ then $\{Q_n = \gamma \times Q_p$ and save action of $S_n =$ opposite action of$"a_i"\}$
   }
   Until $L_i = 1$ for all i;

**Space-Complexity:** In classical Q-learning, if we have n states and m action per state, then the Q-table has a dimension $(m \times n)$. In the conditional Q-learning, we only store Q-value at a state for the best action. Further, we need to store whether the Q-value is already stable or changing. Naturally, we need to store the status of lock variable for each state. Consequently, for each state we require 3 storages, such as Q-value, lock and the best action at that state. Thus for n number of states, we require $(3 \times n)$ number of storage. The saving in memory in the present context with respect to classical Q thus is given by $mn - 3n = n(m-3)$.

## 4 Experiments

Experiment on a large maze of 200×200 show that the run time for planning is less and the optimal path obtained in classical Q-learning is not lost by the present scheme. Snapshots of the planning steps realized on Khepera II are given in Fig. 1 below. The extended Q-learning algorithm is used in the first phase to learn the movement steps from each grid in the map to its neighbor. After the learning phase is over, the planning algorithm is executed with the snapshots as indicated in the Fig.1 to reach the goal.

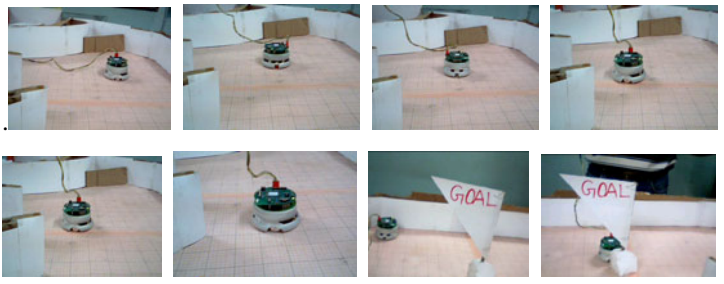


**Fig. 1.** Snapshots of experimental planning instances

## 5 Conclusions

The paper presented an extension of classical Q-learning algorithm, so as to reduce both space and time-complexity by economically updating the Q-table, only when such updating is mandatory. The economic selection of specific entries of the Q-table is performed by certain properties of the extended Q-learning algorithm. The claim of this paper is four fundamental properties, and utilization of these properties to set conditional checking on updating of Q-table entries. This is realized in a pseudo code given in the paper. The reduction in both space-and time-complexity is indicated. Verification of the algorithm has been performed on Khepera II platform. The explanation of the algorithm cannot be included for space limitation.

## References

1. Dean, T., Basye, K., Shewchuk, J.: Reinforcement learning for planning and Control. In: Minton, S. (ed.) Machine Learning Methods for Planning and Scheduling. Morgan Kaufmann, San Francisco (1993)
2. Bellman, R.E.: Dynamic programming, p. 957. Princeton University Press, Princeton (1957)
3. Watkins, C., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)
4. Konar, A.: Computational Intelligence: Principles, Techniques and Applications. Springer, Heidelberg (2005)
5. Busoniu, L., Babushka, R., Schutter, B.D., Ernst, D.: Reinforcement Learning and Dynamic Programming Using Function Approximators. CRC Press, Taylor & Francis Group, Boca Raton, FL (2010)
6. Chakraborty, J., Konar, A., Jain, L.C., Chakraborty, U.: Cooperative Multi-Robot Path Planning Using Differential Evolution. Journal of Intelligent & Fuzzy Systems 20, 13–27 (2009)
7. Gerke, M., Hoyer, H.: Planning of Optimal paths for autonomous agents moving in in homogeneous environments. In: Proceedings of the 8th Int. Conf. on Advanced Robotics, pp. 347–352 (July 1997)
8. Xiao, J., Michalewicz, Z., Zhang, L., Trojanowski, K.: Adaptive Evolutionary Planner/Navigator for Mobile robots. IEEE Transactions on Evolutionary Computation 1(1) (April 1997)
9. Bien, Z., Lee, J.: A Minimum–Time trajectory planning Method for Two Robots. IEEE Trans. on Robotics and Automation 8(3), 443–450 (1992)