A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot

Amit Konar, *Senior Member, IEEE*, Indrani Goswami Chakraborty, Sapam Jitu Singh,
Lakhmi C. Jain, and Atulya K. Nagar

***Abstract*—This paper provides a new deterministic Q-learning with a presumed knowledge about the distance from the current state to both the next state and the goal. This knowledge is efficiently used to update the entries in the Q-table once only by utilizing four derived properties of the Q-learning, instead of repeatedly updating them like the classical Q-learning. Naturally, the proposed algorithm has an insignificantly small time complexity in comparison to its classical counterpart. Furthermore, the proposed algorithm stores the Q-value for the best possible action at a state and thus saves significant storage. Experiments undertaken on simulated maze and real platforms confirm that the Q-table obtained by the proposed Q-learning when used for the path-planning application of mobile robots outperforms both the classical and the extended Q-learning with respect to three metrics: traversal time, number of states traversed, and 90° turns required. The reduction in 90° turnings minimizes the energy consumption and thus has importance in the robotics literature.**

***Index Terms*—Agent, mobile robots, path planning, Q-learning, reinforcement learning.**

I. 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 given criteria [8], [9]. However, the environment may be imprecise, vast, dynamical, and partially nonstructured [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 planning scheme in a cluttered environment, the controller of such kind of robots must be adaptive in nature. 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 an offline algorithm. It is called online if it is capable of producing a new path in response to environmental changes.

Machine learning is generally employed in a mobile robot to make it aware about its world map. The early research on mobility management of robots emphasized the needs of supervised learning to train a robot to determine its next position in a given map using the sensory readings obtained by the robot about the environment. Supervised learning is a good choice for mobility management of robots in fixed maps. However, if there is a small change in the robot's world, the acquired knowledge is no longer useful to guide the robot to select its next position. A complete training of the robot with both the old and the new sensory data–action pairs is then required to overcome the said problem.

Reinforcement learning is an alternative learning policy, which rests on the principle of reward and punishment. No prior training instances are presumed in reinforcement learning. A learning agent here does an action on the environment and receives a feedback from the environment based on its action. The feedback provides an immediate reward for the agent. The learning agent here usually adapts its parameter based on the current and cumulative (future) rewards. Since the exact value of the future reward is not known, it is guessed from the knowledge about the robot's world map. The primary advantage of reinforcement learning lies in its inherent power of automatic learning even in the presence of small changes in the world map.

Usually, the planning involves an *action policy* to reach a desired goal state, through the maximization of a *value function* [1]–[3], which designates subobjectives and helps in choosing the best path. For instance, the value function could be the shortest path, the path with the shortest time, the safest path, or any combination of different subobjectives. The definition of a task in this context may contain, besides the value function, some *a priori* knowledge about the domains, such as the environmental map, the environmental dynamics, and the goal position. The *a priori* knowledge helps the robot in generating a plan for motion amid obstacles, while the lack of such knowledge obliges the robot to learn it first before invoking the motion planning algorithm.

Our research applies reinforcement learning techniques to real-world robots. Reinforcement learning has been tested in many simulated environments [3]–[11] but on a limited basis in real-world scenarios. A real-world environment poses more challenges than a simulated environment, such as enlarged state spaces [2], increased computational complexity, significant safety issues (a real robot can cause real damage), and

Manuscript received June 17, 2011; revised February 22, 2012; accepted September 29, 2012. Date of publication August 1, 2013; date of current version August 14, 2013. This paper was recommended by Associate Editor R. Roberts.

A. Konar, I. Goswami Chakraborty, and S. J. Singh are with the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700032, India (e-mail: konaramit@yahoo.co.in; ic_iit@yahoo.com; sapamjitu@yahoo.com).

L. C. Jain is with the University of South Australia, Adelaide, S.A. 5000, Australia (e-mail: Lakhmi.Jain@unisa.edu.au).

A. K. Nagar is with the Liverpool Hope University, Liverpool L16 9JD, U.K. (e-mail: nagara@hope.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2012.2227719

longer turnaround times for results. This research measures how well reinforcement-learning technique, such as Q-learning, can be applied to the real robot for navigational problem. In our research, we modified the classical Q-learning (CQL) algorithm, hereafter called the improved Q-learning (IQL) for increasing its performance in the path-planning problem.

The performance of a reinforcement-learning algorithm is greatly influenced by two important factors used in the control strategy of the algorithm, popularly known as "exploration" and "exploitation." Exploration usually refers to selecting any action with nonzero probability in every encountered state to learn the environment by the agent. Exploitation, on the other hand, is targeted at employing the current knowledge of the agent to expect achieving good performance by selecting greedy actions [37]. One classical method to balance exploration and exploitation in Q-learning is ε-greedy exploration [39], where a parameter ε representing exploration probability is introduced to control the ratio between exploration and greedy action selection.

The second alternative method to handle the aforementioned problem is to employ the Boltzmann exploration, where the agent selects an action a with a probability: $\exp(Q(S, a)/T/ \sum_{a/} \exp Q(S, a')/T$ in which Q (S, a) denotes the Q-value at state S due to action a and T is a positive constant called the "temperature parameter." The Boltzmann exploration ensures that the agent is more likely to select action a with higher Q-value: Q (S, a). The temperature parameter T adjusts the balance between exploration and exploitation. Large T refers to better exploration, while small T approaching zero indicates almost deterministic (greedy) action selection. Usually, these parameters are determined by trial and error to achieve desired performance.

In [38], the authors employed an interesting strategy of internal prediction/estimation to control the balance between exploration and exploitation for the efficient adaptability of an agent in a new environment. Here, a reliability index (RI) has been introduced as an internal variable to estimate the "expected prediction error." The variable RI is updated after every action by comparing itself to the actual error. The RI is introduced in the expression of the Boltzmann exploration by replacing temperature parameters T by $RI = R(S)/\eta$, where η is a positive constant and S denotes the current state of the agent. The factor η is used to normalize the magnitude of Q(s, a) with R(s).

In the context of path planning by a mobile robot, the robotic agent usually has additional knowledge about the distance from the current state to both the next state and the goal. This knowledge has been efficiently used here during the learning phase of the robotic planner to speed up learning through greedy selection of actions. Four properties (rules) concerning the computation of Q-values at state $S^{/}$ from the current estimate of the Q-value at state S, where $S^{/}$ is a neighbor of state S, have been developed. Each rule has a conditional part and an action part. If the conditional part is found true, the action part is realized. The conditional part involves checking a locking status of a state along with a distance comparison, where the locking of a state indicates that its Q-value needs no further updating. The action part ensures that the Q-value at $S^{/}$ can be evaluated in one step only, and the state $S^{/}$ will also be locked. Thus, when the conditional part of a property is activated, the Q-value at a new state is evaluated once only for ever, and the new state is locked. The proposed IQL algorithm is terminated when all the states in the workspace are locked.

It is apparent from the aforementioned discussion that exploration in the IQL algorithm takes place when all the rules' conditional parts do not satisfy at a given situation, whereas exploitation takes place when the conditional part of one rule is activated. The balancing of exploration and exploitation is done naturally, and no parameter is involved to control balancing.

This paper is an extension of the extended Q-learning (EQL) [17] algorithm, where, too, the authors presumed that their learning algorithm has knowledge about the distance measure of the current state to the next state and the goal. They also listed four properties (rules) of EQL without proof, two of which are derived in this paper. However, the remaining two properties presented here are novel. These two new properties provide alternative conditions for locking, thereby improving the relative learning speed of the proposed algorithm in comparison to EQL.

The work presented in this paper is better than the CQL [3] and the EQL by the following counts.

1) In CQL, the Q-values of the states are updated theoretically for infinite number of steps. For all practical purposes, however, the algorithm is terminated when the difference in the Q-values of each state in two successive iterations is within a prescribed limit. The algorithm is said to have converged under this circumstance, which too requires excessive computational time. The time complexity of the proposed Q-learning algorithm has been reduced here by locking selected states, where the Q-value update is no longer required. The conditions used for identifying the states to be locked are derived here.
2) The EQL presented in [17] stores only the best action at a state. Naturally, the knowledge acquired by Q-learning in a world map without obstacles cannot be correctly used for planning, particularly when the next state due to the best action is occupied with an obstacle.

   In the modified Q-learning presented here, the agent is capable of ranking all the actions at a state based on the Q-values at its neighboring states. Consequently, during the planning cycle, if the state corresponding to the best action is occupied with an obstacle, the robot would pick up the next best action. This, in one way, overcomes one fundamental limitation of the EQL.
3) Since the EQL stores only the best action at a state, it cannot take care of the multiplicity of the best actions. Thus, if there exist two or more best actions, it selects one of them arbitrarily and saves the selected action with the state. Naturally, the stored action may sometimes involve more turning of the robot during planning than it could have been obtained by an alternative best action. In our algorithm, if we have more than one competitive action at a state during the planning cycle, we select the one ensuring minimum turning of the robot. Thus, our present algorithm is energy optimal.
4) Both the EQL [17] and IQL algorithms are terminated when all the states are locked. However, the IQL has

four locking conditions, including the two of the EQL. Because of these two additional conditions of locking, the probability of locking of a state in a given interval of time by the IQL is higher than the EQL.

The rest of this paper is organized as follows. CQL is introduced in Section II. Some properties of the Q-learning based on the concept of locking of states are derived in Section III. The algorithm for the IQL is given in Section IV. The algorithm for the path planning is given in Section V. Computer simulation is shown in Section VI. Experimental details are included in Section VII. Conclusions are listed in Section VIII.

II. CQL

In CQL, all possible states of an agent and its possible actions in a given state are deterministically known. In other words, for a given agent A, let $S_1, S_2, \ldots, S_n$ be n possible states, where each state has m possible actions $a_1, a_2, \ldots, a_m$. At a particular state–action pair, the specific reward that the agent acquires is known as *immediate reward*. Let $r(S_i, a_j)$ be the immediate reward that agent A acquires by executing an action a_j at state S_i. The agent selects its next state from its current states by using a policy. The policy attempts to maximize the cumulative reward that the agent could have in the subsequent transition of states from its next state. Let the agent be in state S_i, and it is expecting to select the next best state. Then, the Q-value at state S_i due to the action of a_j[36] is given in

$$Q(S_i, a_j) = r(S_i, a_j) + \gamma \operatorname*{Max}_{a^/} Q\left(\delta(S_i, a_j), a^/\right) \tag{1}$$

where $0 < \gamma < 1$ and $\delta(S_i, a_j)$ denotes the next state due to the 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, the selection of $a^/$ that maximizes $Q(S_k, a^/)$ and, in turn, $Q(S_i, a_j)$ is an interesting problem.

The CQL algorithm for deterministic state transitions is given hereinafter. The algorithm starts with a randomly selected initial state. An action "a" from a list of actions $a_1, a_2, \ldots, a_m$ is selected, and the agent, because of this action, receives an *immediate reward* r and moves to the new state following the δ-transition rule given in a table. The Q-value of the previous state due to the action of the agent is updated following the Q-learning (1). Now, the next state is considered as the initial state, and the steps of action selection, receiving *immediate reward*, transition to the next state, and Q-update are represented for ever.

```
**Classical Deterministic Q-Learning**
**For** each $S, a$ initialize $Q(S, a) = 0$;
Observe the current state $S$;
**Repeat**
    Select $a \in \{a_1, a_2, \ldots, a_m\}$ and execute it;
    Receive an immediate reward $r(S, a)$;
    Observe the new state $S^/ \leftarrow \delta(S, a)$;
    Update the table entry $Q(S, a)$ by
        $Q(S, a) = r(S, a) + \gamma \operatorname*{Max}_{a^/} Q(\delta(S, a), a^/)$;
    $S \leftarrow S^/$;
**For ever**.
```

The CQL requires a memory of $(n \times m)$ to keep track of the Q-table. For large n and m, the space complexity thus is high. In the IQL, we attempted to reduce the space complexity. The IQL algorithm presented here involves n Boolean variables called Lock for n states to indicate whether $Q(S, a)$ at state S due to action a needs to be updated. The Lock variables are used to avoid unnecessary update of entries $Q(S, a)$ in the Q-table and, thus, to save time complexity.

In IQL, we require n-memories to store n-Lock variables associated with n states. Here, instead of the Q-table of $n \times m$ dimension, we require to store the best Q-value of a state because of any action and thus require n-memories for n best Q-values of n states. Therefore, we save some space complexity $(\mathrm{nm} - 2\,\mathrm{n}) = \mathrm{n}(\mathrm{m} - 2)$.

III. Properties of the IQL

This section provides four interesting properties, based on which the IQL algorithm has been developed. The properties stress upon the following two issues.

1) If the Q-value at any state S (is known and) needs no updating, the state is locked.
2) If S is a locked state, $S^/$ is a neighboring state of S, and any one of the four properties to be derived is applicable at the (S, $S^/$) pair, then we can compute the Q-value at state $S^/$ once only using the property, and the $S^/$ is also locked.

If no property is applicable at a given (S, $S^/$) pair, state transition takes place without any updating in the Q-value. It may not be out of place to mention here that the first locking in IQL takes place when the agent has a state transition from the goal state to any of the neighboring states of the goal. The locking of states then is continued as and when one of the four properties is applicable. The IQL terminates when all the states are locked. We now formally define some parameters to derive the properties.

Let, for any state S_k, the distance between the goal state and the next feasible states of S_k be known. Let the next feasible state of S_k be $S \in \{S_a, S_b, S_c, S_d\}$. Let G be the goal and the city block distances between S_a, S_b, S_c, S_d, and G be d_{aG}, d_{bG}, d_{cG}, and d_{dG}, respectively. Let the distance, in order, be $d_{\mathrm{bG}} < d_{\mathrm{aG}} < d_{\mathrm{cG}} < d_{\mathrm{dG}}$. 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:

$$\begin{aligned} Q(S_k, a^/) &= r(S_k, a^/) + \gamma \operatorname*{Max}_{a^{//}} Q\left(\delta(S_k, a^/), a^{//}\right) \\ &= 0 + \gamma \operatorname*{Max}_{a^/} Q\left(\delta(S_k, a^/), a^{//}\right). \end{aligned} \tag{2}$$

Now, $\delta(S_k, a^/) = S_a|S_b|S_c|S_d$, where | denotes the OR operator.

Therefore

$$\begin{aligned} &\operatorname*{Max}_{a^{//}} Q\left(\delta(S_k, a^/), a^{//}\right) \\ &\quad = \operatorname*{Max}_{a^{//}} Q\{S_a|S_b|S_c|S_d, a^{//}\} \\ &\quad = Q(S_b, a^{//}).(\because d_{bG} < d_{aG} < d_{cG} < d_{dG}). \end{aligned} \tag{3}$$

Combining (2) and (3), we have

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

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 the $\gamma \times Q$-value of the next state.

Let S_p, S_n, and S_G be the present, the next, and the goal states, respectively. Let Q_p and Q_n be the Q-value at the present and next states S_p and S_n, respectively, for the best action. Let d_{xy} be the city block 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 will not change further after L_n is set to 1. The Lock variable for all states except the goal will be initialized to zero in our proposed Q-learning algorithm. We observe four interesting properties as indicated hereinafter.

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

Proof: Let the neighborhood state of S_p be $S \in \{S_a, S_b, S_c, S_n\}$, and the agent selects S_n as the next state as $d_{nG} < d_{xG}$ for $x \in \{a, b, c, n\}$.

Now

$$\begin{aligned} Q_p &= Q(S_p, a) \\ &= r(S_p, a) + \gamma \operatorname*{Max}_{a^{/}} Q\left(\delta(S_p, a)a^{/}\right) \\ &= 0 + \gamma \operatorname*{Max}_{a^{/}} Q(S_a|S_b|S_c|S_n, a^{/}) \\ &= \gamma \times Q(S_n, a^{/})\ (\because\ d_{nG} \le d_{xG}, \forall x) \\ &= \gamma \times Q_n. \end{aligned} \tag{4}$$

Since $L_n = 1$ and $d_{pG} > d_{nG}$, $\therefore Q_p < Q_n$, and thus, $Q_p = \gamma \times Q_n$ for $0 < \gamma < 1$ is the largest possible value of Q_p, so Q_p should not be updated further. Therefore, $L_p = 1$ is set. □

Property 2: If $L_p = 1$ and $d_{nG} < d_{pG}$, then $Q_n = Q_p/\gamma$ and set $L_n = 1$.

Proof: Since $d_{nG} < d_{pG}$, the agent will select the next state n from the current state p. Hence, by (4)

$$Q_p = \gamma \times Q_n \Rightarrow Q_n = \frac{Q_p}{\gamma}. \tag{5}$$

Since $L_p = 1$ and $d_{nG} < d_{pG}$, $\therefore Q_p < Q_n$, and thus, $Q_n = Q_p/\gamma$ for $0 < \gamma < 1$ is the largest possible value of Q_n, so Q_n should not be updated further. Therefore, $L_n = 1$ is set. □

Property 3: If $L_p = 1$ and $d_{nG} > d_{pG}$, then $Q_n = \gamma \times Q_p$ and set $L_n = 1$.

Proof: If the robot is moving from S_i to the goal G in one step, the immediate reward is R, for example. On the other hand, if the robot moves from S_i to any state other than the goal G, then the immediate reward is zero.

Now, suppose that the robot moves from S_p to the goal G in k. Now, as $L_p = 1$, k is the minimum number of state transitions to reach the goal from S_p. Therefore, we obtain

$$\begin{aligned} Q_p &= Q(S_p, a) \\ &= r(S_p, a) + \gamma \operatorname*{Max}_{a^{/}} Q\left(\delta(S_p, a), a^{/}\right) \\ &= 0 + \gamma^k R. \end{aligned} \tag{6}$$

Suppose that the robot moves from S_n to the goal G in $k + 1$ transition steps. Here, $(k + 1)$ is also the minimum number of steps to reach the goal G from S_n, failing which the agent would have selected some other state as the next state from the current state S_p

$$\begin{aligned} Q_n &= Q(S_n, a) \\ &= r(S_n, a) + \gamma \operatorname*{Max}_{a^{/}} Q\left(\delta(S_n, a), a^{/}\right) \\ &= 0 + \gamma^{k+1} R. \end{aligned} \tag{7}$$

Dividing (6) by (7), we have

$$\begin{aligned} \frac{Q_p}{Q_n} &= \frac{\gamma^k R}{\gamma^{k+1} R} \\ &= \frac{1}{\gamma} \\ \Rightarrow Q_n &= \gamma \times Q_p. \end{aligned} \tag{8}$$

Since $d_{nG} > d_{pG}$, $Q_n < Q_p$. Therefore, $Q_n = \gamma \times Q_p$ has the largest possible value for $0 > \gamma > 1$. Furthermore, as $L_p = 1$ and S_n is the nearest state to S_p with respect to the given distance metric, therefore, L_n is set to 1. □

Property 4: If $L_n = 1$ and $d_{nG} > d_{pG}$, then $Q_p = Q_n/\gamma$ and set $L_p = 1$.

Proof: Since $d_{nG} > d_{pG}$, Q_n can be evaluated from Q_p by (8). Thus

$$Q_n = \gamma \times Q_p \Rightarrow Q_p = Q_n/\gamma. \tag{9}$$

Since $L_n = 1$ and $d_{nG} > d_{pG}$, $\therefore Q_n < Q_p$, and thus, $Q_p = Q_n/\gamma$ for $0 < \gamma < 1$ is the largest possible value of Q_p, so Q_p should not be updated further. Therefore, $L_p = 1$ is set. □

IV. IQL Algorithm

The CQL employs a Q-table to store the $Q(S, a)$ for $S = S_1$ to S_n and $a = a_1$ to a_m. Thus, it requires an array of $(n \times m)$ size. In the IQL, we, however, require to store only the Q-values at a state S for the best action. Thus, for n states, we need to store n Q-values. Aside from the Q-storage, we, in addition, require n Boolean Lock variables, denoted by L_i for state S_i, $i = 1$ to n, depicting the current status of the state. If the Lock variable at a state is 1, then the Q-value at that state need not be updated further.

In the path-planning application of mobile robots, the environment can be partitioned into nonoverlapped grids, called states. Thus, a state can have four neighbors. Consequently, during the planning phase, the robot can determine the best action to move to the next (best) state S_n from the current S_p by identifying the neighboring state having the largest Q-value.

The corresponding action of the robot to move to the next (best) state S_n is apparent.

The proposed algorithm for IQL has two main steps: 1) initialization and 2) Q-table updating. In the initialization phase, the lock variable at all states except the goal state S_G is set to zero. The immediate reward from any neighboring state to the goal state is set to 100. The discounting factor γ and the initial state are fixed up.

In the present update policy of the Q-table, if $L_p(L_n)$ is 1, then $L_n(L_p)$ will be set to 1. However, in the initialization phase, only L_G is set to 1. Thus, unless the current or the next state $= L_G$, there will be no update in the Q-table. In order to have L_p or $L_n = L_G$, the robot usually has to wander in its world map for a finitely large number of iterations. To avoid the unnecessary execution of the Q-table update, we add a small repeat-until loop between the two main phases of the program. This loop continues selecting an action and executing it (without updating the Q-table) until the robot reaches the goal.

Once the robot reaches the goal, the first repeat-until loop exits, and the Q-table updating is initiated. The process of Q-table updating is continued until all the states are locked.

The pseudocode of the IQL is given hereinafter.

Pseudocode for IQL

```
1. Initialization
   For all $S_i$, $i = 1$ to $n$, except $S_i = S_G$
   {set $L_i = 0$; $Q_i = 0$;}
   $L_G$(for goal $S_G$) $= 1$;
   $Q_G$(for goal $S_G$) $= 100$;
   Assign $\gamma$ in (0, 1) and initial state $= S_p$;
2. Repeat
   {
      Select $a_i$ from $A = \{a_1, a_2, \ldots, a_m\}$ and execute it;
   } Until$S_P = S_G$;
3. Update Q-table:
Repeat
   {
      a) Select $a_i$ from $A = \{a_1, a_2, \ldots, a_m\}$;
      b) Determine $d_{nG}$ and $d_{pG}$;
            If$(d_{nG} < d_{pG})$
            Then if$(L_n = 1)$
                  Then if$(L_p = 0)$
                        Then $\{Q_p = \gamma \times Q_n; L_p = 1;\}$
                  Else if$(L_p = 1)$
                        Then $\{Q_n = Q_p/\gamma; L_n = 1;\}$
            Else if$(L_p = 1)$
                  Then if$(L_n = 0)$
                        Then $\{Q_n = \gamma \times Q_p; L_n = 1;\}$
                  Else if$(L_n = 1)$
                        Then $\{Q_p = Q_n/\gamma; L_p = 1;\}$
   } Until$L_i = 1$ for all $i$ without obstacle;
```

Theorem 1: The entries in the Q-table for the best action in the CQL have the same value as that in the IQL.

Proof: Let Q_p and Q_n be the Q-values for the best action at the present state S_p and the next state S_n, respectively. In CQL, Q_p is updated when the agent has a transition from S_p to S_n. However, Q_p would attain the maximum value in a learning epoch if the Q_n had already attained the maximum value.

Now, if S_p is closest to the goal, then $Q_p = \gamma.R$, where R is the immediate reward and γ is the discounting factor. Therefore, if S_p is at a distance of k through a shortest path measured by the city block distance, then $Q_p = \gamma^k.R$. Q_p, if updated later, cannot exceed $\gamma^k.R$, as it is at a shortest distance k w.r.t. the goal.

In the IQL, if $d_{pG} > d_{nG}$, then, by property (1) and (2), $Q_p = \gamma.Q_n$. Now, if $L_n = 1$, i.e., state n is at a shortest distance $(k-1)$ to the goal, $Q_n = \gamma^{k-1}.R$, and then, $Q_p = \gamma.Q_n = \gamma.\gamma^{k-1}.R = \gamma^k.R$. Therefore, when $d_{pG} > d_{nG}$ and $L_n = 1$, $Q_p = \gamma^k.R$, and this value of Q_p should not change further.

Furthermore, if $d_{nG} > d_{pG}$, then, by property (3) and (4), $Q_p = Q_n/\gamma$. Now, if Q_p is at a shortest distance $(k+1)$ from the goal, then $Q_n = \gamma^{k+1}.R$, and $\therefore Q_p = Q_n/\gamma = \gamma^k.R$; $\gamma^k.R$ is the largest possible value of Q_p.

In the IQL, irrespective of $d_{pG} > d_{nG}$ or $d_{nG} > d_{pG}$, it is found that $Q_p = \gamma^k.R$. Therefore, both the CQL and the IQL have a steady-state Q-table with $Q_p = \gamma^k.R, \forall p$. □

We now determine the space and the time complexity of the IQL.

Space Complexity: In CQL, if there are n states and m actions per state, then the Q-table will be of $(m \times n)$ dimension. In the IQL, two storages are required for each state, one for storing the Q-value and the other for storing the value of the lock variable of a particular state. Thus, for n number of states, we require a Q-table of $(2 \times n)$ dimension. The saving in memory in the present context with respect to classical Q thus is given by mn − 2 n = n(m − 2), which is of the order of mn.

Time Complexity: In CQL, the updating of Q-values in a given state requires determining the largest Q-value in the next state for all possible actions by (1). Thus, if there are m possible actions at a given state, the maximization of m possible Q-values requires $m-1$ comparison. Consequently, if we have n number of 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. However, we need to know whether state n is locked, i.e., the Q-value of S_n is permanent and stable. Thus, if we have n number of states, we require n number of comparisons. Consequently, we save a time n(m − 1) − n = nm − 2 n = n(m − 2), which is of the order of mn.

V. Path-Planning Algorithm

The Q-learning algorithm presented earlier stores the Q-values at each state for the best action. After the learning is completed, i.e., all the states are locked, the Q-table can be used for path-planning application. During path planning, the robot, while at state S_p, identifies the next best state S_n, where the Q-value is higher than the Q-value of other neighboring states of S_p. However, if there exist more than one next state having the largest Q-value among the neighboring state of S_P, the robot ideally would select any one of them.

In this paper, we, however, economically select the next state S_n, while the robot is at S_p, based on the torque requirement. For example, let there exist two states S_{n1} and S_{n2} having the largest Q-value around the neighbor of S_p. Then, the robot would select S_{n1} as the next state if the angular rotation to move to S_{n1} is smaller than that of S_{n2}. A small angular turning requires less torque to be generated by the robot. Consequently, the robot in the proposed planning algorithm consumes minimum energy as the torques generated during successive movement of the robot toward the goal is optimally selected.

Let the present state be S_p and S_n be the next state with the largest Q-value. Let S_r be a next state of S_p and S_G be the goal state. We now develop a path-planning algorithm to determine an obstacle-free trajectory of optimal path length and energy for the robot between an arbitrary starting point and a fixed goal point in the pretrained world map.

Pseudocode for Path Planning

```
BEGIN
   1. $CURRENT \leftarrow S_p$;
   2. If $Q_n > Q_r \; \forall r$
          Then, $\forall n$, select $n^{/}$ from $n$ such that the angular rotation required to reach $n^{/}$ is minimum and $n^{/}$ is obstacle free;
   3. Go to $NEXT$;
   4. $CURRENT = NEXT$;
   5. Repeat from step 2 until $NEXT = GOAL$;
END
```

VI. Computer Simulation

In our computer simulation, we consider an environment of 20 × 20 grids, where each grid is given a state number. For example, a grid located at position (x, y) defined in the Cartesian coordinate reference frame has a

$$stateno. = (x - 1) \times rowsize + y \tag{10}$$

where $rowsize$ denotes the number of grids in a row.

Performance of the Q-learning algorithm has been studied here in two phases. First, a given world map is trained by the proposed Q-learning algorithm. Second, the trained world map with a known Q-table is used to generate a trajectory of motion of the robot between an arbitrarily selected initial position and the fixed goal position in the said world map. The performance metrics used here to compare the relative performance of our proposed algorithm with the CQL and the EQL [5] include the convergence time of the learning algorithm, total time taken to execute a plan of motion in a pretrained world map, and the number of ±90° rotations involved to completely execute the plan. While convergence is considered for the learning algorithm, the issues of time and energy consumptions, the latter being measured in terms of ±90° turnings, are part of the planning algorithm. The performance analysis considers both the learning and the planning algorithms together as both of them jointly determine the overall performance in the path-planning application.

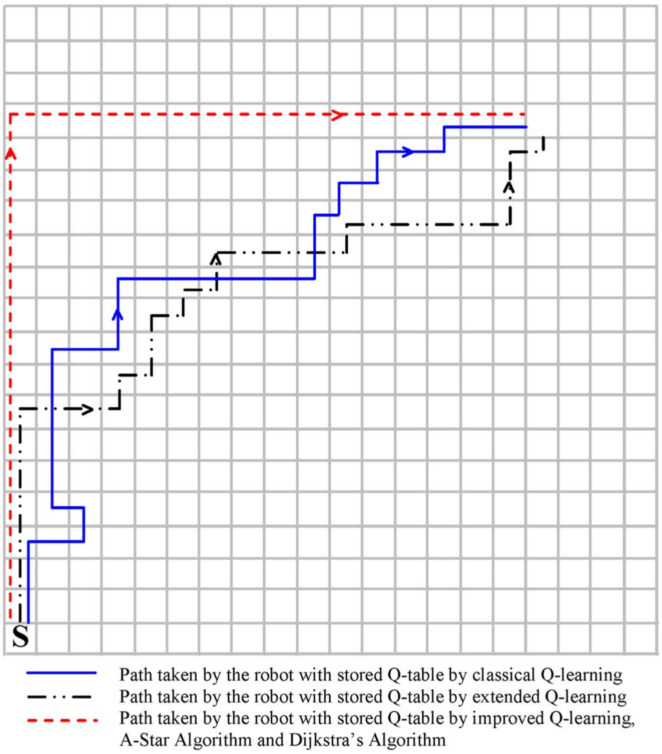
S
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning, A-Star Algorithm and Dijkstra's Algorithm

Fig. 1. World map 1 without obstacle. Paths taken by the robot with stored Q-table by the different algorithms are shown in the figure.

The performance of the Q-learning algorithm is studied under four experimental settings. First, the training and planning are performed on the same obstacle-free world maps. Second, the training is performed on an obstacle-free map, but before the planning algorithm is executed, obstacles are added in the map. Third, both training and planning are performed on the same map with few obstacles. Finally, the training was performed in a map with few obstacles, and a few more obstacles are added before planning. The CQL, EQL, and IQL algorithms are compared with the well-known Dijkstra's shortest path finding algorithm in a graph and the heuristic A^* algorithm under the aforementioned four experimental settings.

A. Experiments

Let S and G be the starting and the goal states in all the world maps considered for path planning in the following experiments. In each experiment, the Q-table is obtained by three different Q-learning algorithms: the CQL, EQL, and IQL. After the learning is over, the robot is kept at the starting position with the heading direction in the east. The experiments and the corresponding results are briefly outlined hereinafter.

Experiment 1: The first experiment is carried out on a world map of 20 × 20 grids as shown in Fig. 1 to compare the relative performance of the three distinct Q-learning algorithms. The CQL with random action selection and Boltzmann action selection with $T = 0.01$ have been found to converge after 50 026 and 8276 iterations.

The EQL and the IQL are executed until all the Lock variables associated with each state are set to 1. The number of iterations required to learn the world map of Fig. 1 by the

TABLE I
COMPARISON OF TIME TAKEN BY THE ROBOT AND NUMBER OF 90° TURNS REQUIRED BY THE ROBOT

World map	Planning time taken in seconds					No. of 90° turns				
	IQL	*EQL*	*CQL*	*A-star*	*Dijkstra*	*IQL*	*EQL*	*CQL*	*A-star*	*Dijkstra*
Fig. 1	22.25	24.82	26.47	40.25	88.25	2	15	16	2	2
Fig. 2	23.07	-	30.76	42.05	91.24	6	-	20	6	6
Fig 3	13.78	14.17	16.70	27.65	52.60	4	4	10	4	4
Fig. 4	23.40	-	32.95	47.90	93.86	11	-	22	10	10
Fig. 5	31.48	-	47.08	64.01	121.35	15	-	38	8	8
Fig. 6	09.34	10.16	14.77	18.52	36.73	2	6	9	2	2
Fig. 7	24.55	-	37.51	49.47	122.31	5	-	29	5	5
Fig. 8	16.81	-	20.21	32.76	63.87	4	-	12	4	4

IQL	Improved Q-learning
EQL	Extended Q-learning
CQL	Classical Q-learning
-	Goal cannot be reached

EQL was found to be 20 273, whereas the IQL requires 6502 iterations to set all the states locked.

Experiment 1 reveals that the paths obtained by the planning algorithm by acquiring knowledge in the Q-table by all the three algorithms are optimal (having 21 state transitions) with respect to a measure of the city block distance. However, the IQL requires minimum turning and thus consumes minimum energy to execute the complete task of path planning (see Table I). As A^* and Dijkstra's algorithms search optimal paths in real time, they respectively consume almost double and four times the time required for path planning by IQL.

Experiment 2: The second experiment is concerned with training in the obstacle-free world map of Fig. 1 by the three Q-learning algorithms. After the training is over, we add obstacles in the map and change the starting position as indicated in Figs. 2–5, and we execute the respective planning programs. The resulting paths obtained by the planning algorithms with the acquired knowledge stored in the Q-tables by the respective Q-learning algorithm reveal interesting observations. First, the resulting Q-tables obtained by the CQL- and IQL-based techniques help the planning algorithms to construct optimal paths in all the maps of Figs. 2–5. Figs. 2, 4, and 5 exhibit an immature termination of the trajectory of motion by the robot when the Q-table is updated by the EQL. This happens because the EQL algorithm stores only the best action at each state, which sometimes is occupied by an obstacle.

The scenario, however, is different in case of modified Q-learning. In modified Q-learning, the agent during the execution of the plan determines the best neighborhood state having the largest Q-value. If the neighboring state thus selected is occupied by an obstacle, the robot selects the next feasible neighboring state with the largest Q-value and selects it as the next state. Consequently, even with only one neighboring state, which was the previous state of the agent which is unoccupied with an obstacle, the agent can return to that state. Thus, the planning algorithm never gets stuck to a state as in case of the EQL.

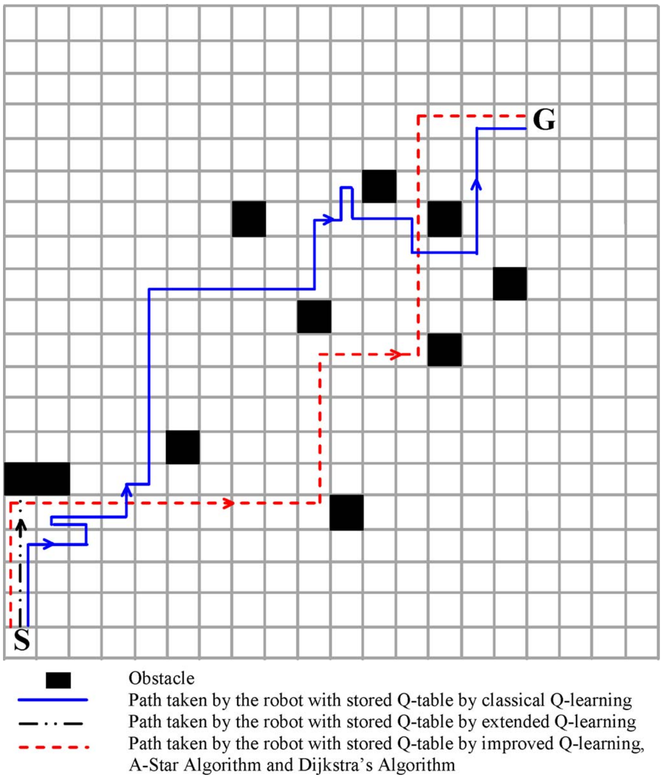
G
S
Obstacle
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning, A-Star Algorithm and Dijkstra's Algorithm

Fig. 2. World map 2 with obstacles. Paths taken by the robot with the stored Q-table by the different algorithms are shown in the figure. The robot with the stored Q-table by the EQL fails to reach the goal.

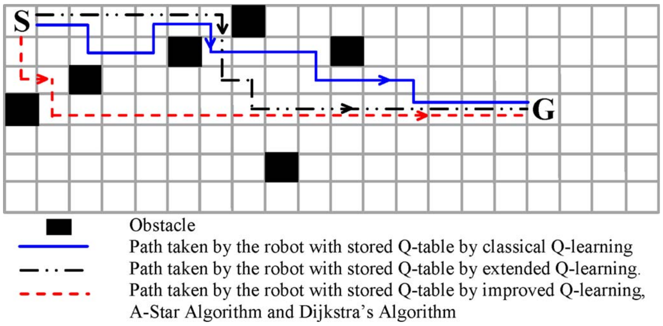
S
G
Obstacle
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning.
Path taken by the robot with stored Q-table by improved Q-learning, A-Star Algorithm and Dijkstra's Algorithm

Fig. 3. Partial World map 3 with obstacles. It shows the paths taken by the robot with the stored Q-table of different algorithms.

It is noteworthy that 90° turns taken by A^* and Dijkstra's algorithms are smaller in comparison to that by IQL in Figs. 4 and 5. The justification of the results is due to the phenomenon that the Q-values stored do not carry information about turning angles. Therefore, a search algorithm looking for a shortest path naturally identifies a trajectory with less turning angles.

Experiment 3: The third experiment is carried out in a world map (see Fig. 6) with obstacles during both the learning and the planning phase. The numbers of learning epochs required for convergence by different algorithms are 50 604 for CQL with random action selection, 9104 for CQL with Boltzmann action selection $(T = 0.01)$, 21 701 for EQL, and 6546 for IQL.

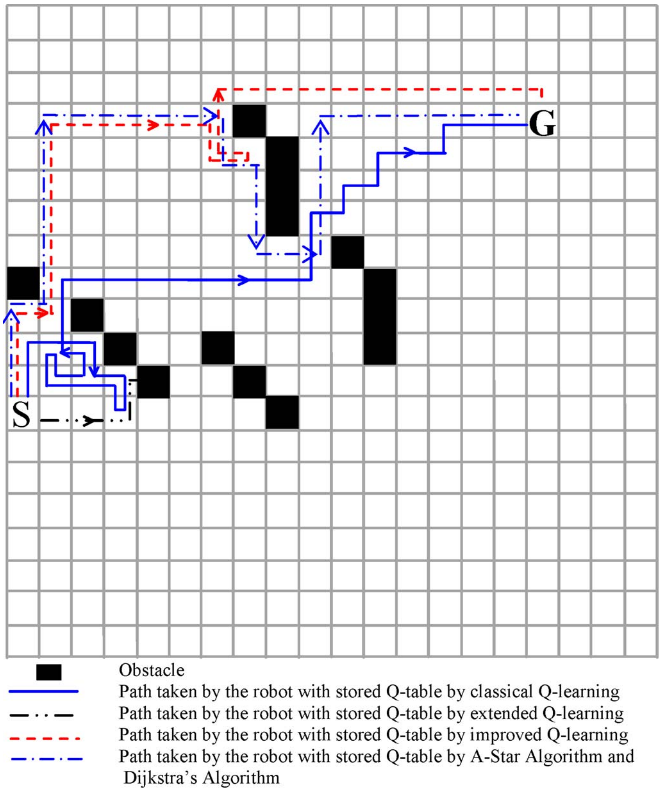
G
S
Obstacle
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning
Path taken by the robot with stored Q-table by A-Star Algorithm and Dijkstra's Algorithm

Fig. 4. World map with obstacles. It shows the paths taken by the robot with the stored Q-table of different algorithms. The robot with the stored Q-table by the EQL fails to reach the goal.

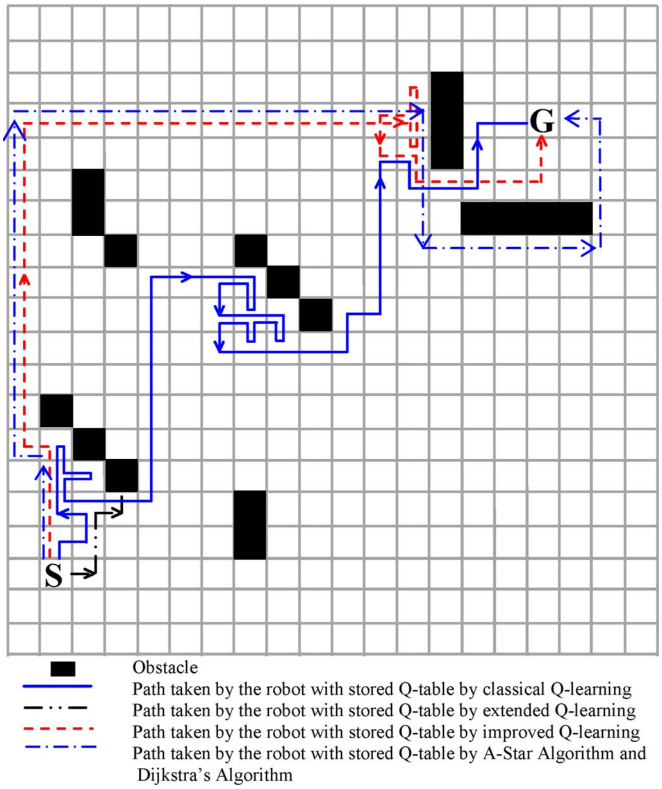
G
S
Obstacle
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning
Path taken by the robot with stored Q-table by A-Star Algorithm and Dijkstra's Algorithm

Fig. 5. World map 5 with obstacles. It shows the path taken by the robot with the stored Q-table of different algorithms. Here, the robot with the stored Q-table by the EQL fails to reach the goal.

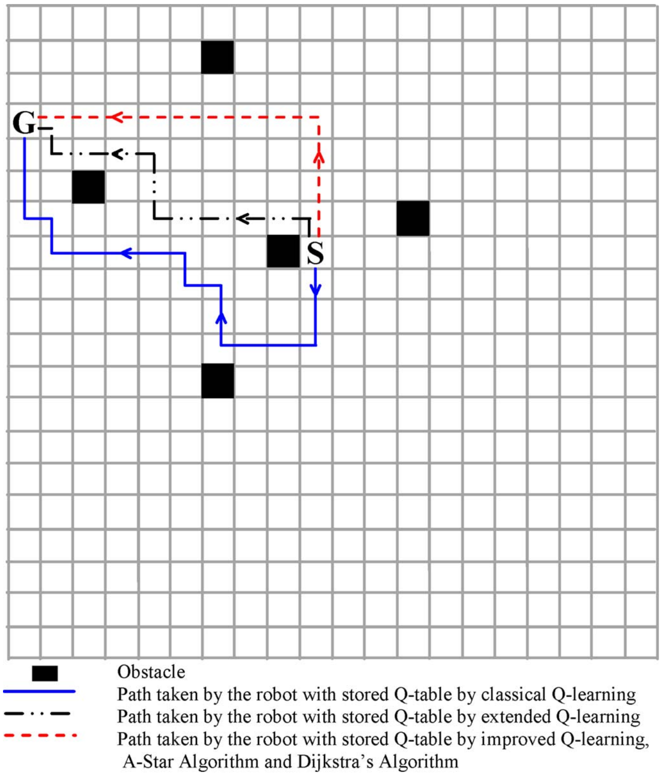
G
S
Obstacle
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning, A-Star Algorithm and Dijkstra's Algorithm

Fig. 6. World map 6 with obstacles. It shows the paths taken by the robot with the stored Q-table of different algorithms.

The planned trajectories obtained by consulting respective Q-tables for CQL, EQL, and IQL algorithms are shown in Fig. 6. The paths planned by Dijkstra's shortest path finding and A^* algorithms are also included in Fig. 6 for comparison. It is apparent from the figure that the shortest path $(\text{no. of state transitions} = 13)$ is obtained when the Q-table is updated both by the EQL and the IQL. On the other hand, the path constructed by consulting the Q-table obtained by the CQL is excessively longer with 19 state transitions. The torque requirement is the smallest in the IQL, as the number of turnings required here is minimum (once only). A^* and Dijkstra's algorithms take excessively large planning time.

Experiment 4: The last experiment is concerned with training in a map with five dark obstacles and planning a trajectory in that map with three additional shaded obstacles (see Fig. 7). It is apparent from Fig. 7 that the IQL- and CQL-induced Q-tables help the robot generate complete trajectories of motion of the robot. However, the planning algorithm realized with the Q-table obtained by the EQL fails to generate a complete trajectory. The number of state transitions and the torque requirement are also minimum in case of IQL. In Fig. 8, we consider five dark obstacles during the training and four additional shaded obstacles during the planning phase. It is noted from the figure that the minimum number of state transitions takes place in case of path planning using the Q-table obtained by IQL. The turning required by the said trajectory is also minimum when compared to the other trajectories.

Figs. 9 and 10 provide information about the number of states locked for EQL and IQL. It is apparent from the figures that IQL takes relatively smaller number of iterations than EQL

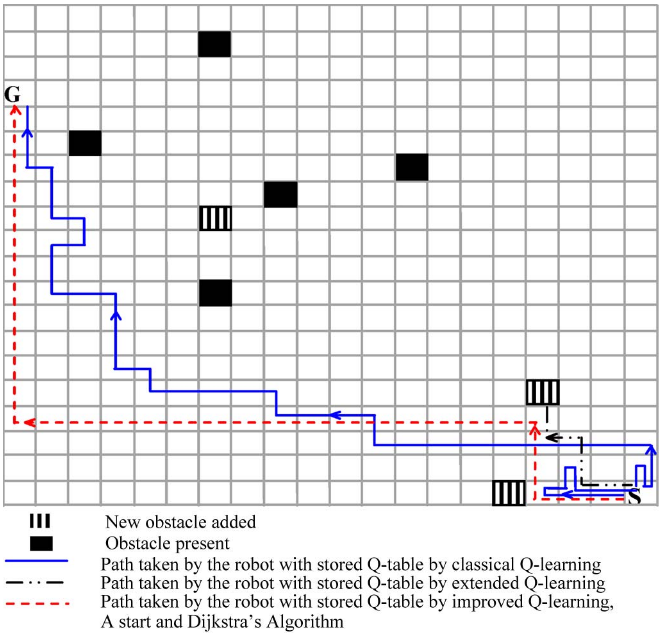
G
S
New obstacle added
Obstacle present
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning, A start and Dijkstra's Algorithm

Fig. 7. World map 7 with obstacles. Paths taken by the robot with stored Q-table by the different algorithms are shown in the figure. Here, the robot with the stored Q-table by the EQL fails to reach the goal.

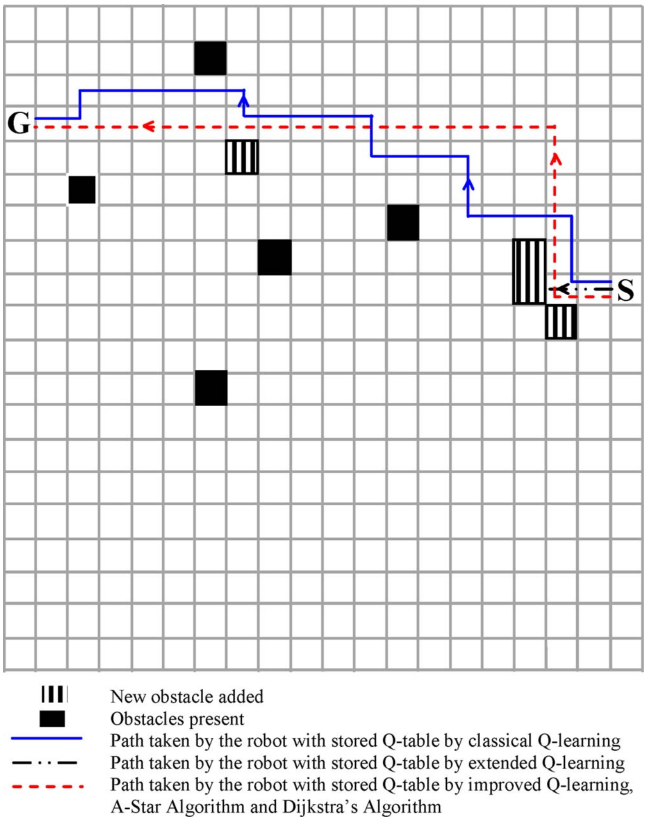
G
S
New obstacle added
Obstacles present
Path taken by the robot with stored Q-table by classical Q-learning
Path taken by the robot with stored Q-table by extended Q-learning
Path taken by the robot with stored Q-table by improved Q-learning, A-Star Algorithm and Dijkstra's Algorithm

Fig. 8. World map 8 with obstacles. Paths taken by the robot with stored Q-table by the different algorithms are shown in the figure. The robot with the stored Q-table by the EQL fails to reach the goal.

for having the same number of states locked. This justifies the significance of the two additional locking conditions in IQL when compared to EQL.

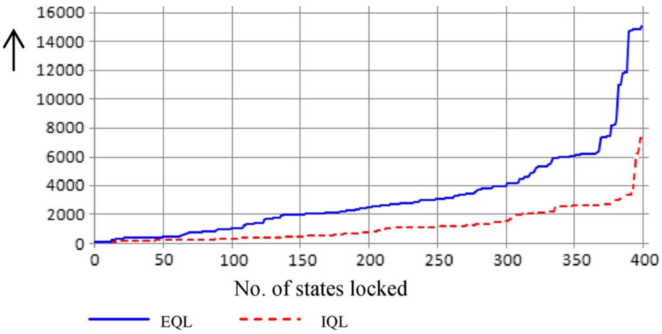

Fig. 9. Comparison between number of iterations required by the IQL algorithm and the EQL algorithm to learn the world map (given in Fig. 2) without any obstacle.

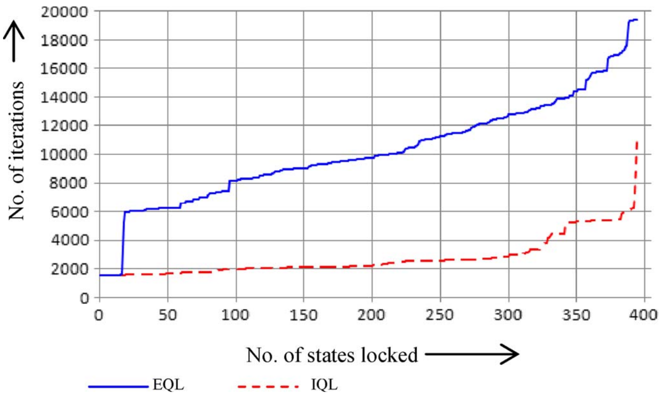

Fig. 10. Comparison between number of iteration required by the IQL algorithm and the EQL algorithm to learn the world map (given in Fig. 7) with five obstacles.

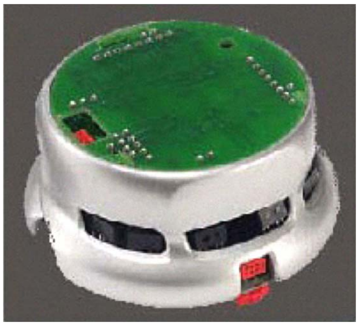

Fig. 11. Khepera II robot.

VII. Experiments With Khepera II Robot

Khepera II (see Fig. 11) is a miniature robot (diameter of 7 cm) equipped with eight built-in infrared proximity sensors and two relatively accurate encoders for the two motors. The range sensors are positioned at fixed angles and have limited range detection capabilities. The sensors are numbered between 0 and 7 with the leftmost sensor designated by 0 and the rightmost one designated by 7 (see Fig. 12). The robot represents measured range data in the scale: [0, 1023]. When an obstacle is away from the sensor by more than 5 cm, it is represented by zero. When an obstacle is approximately 2 cm away, it is represented by 1023. The onboard microprocessor includes

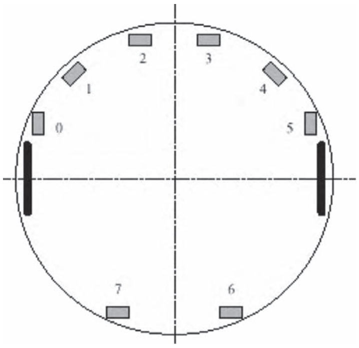

Fig. 12. Position of the sensors of Khepera II.

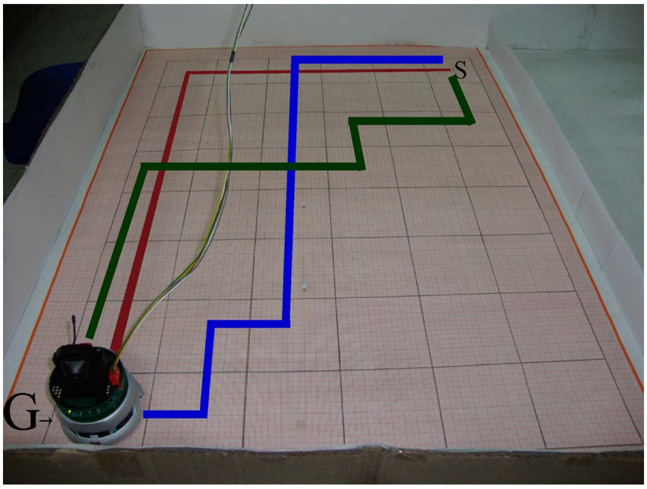

Fig. 13. World map without any obstacle. Path planned by the robot: 🟥 Using the Q-table returned by the IQL. 🟦 Using the Q-table returned by the CQL. 🟩 Using the Q-table returned by the EQL.

a flash memory of 512 KB and a Motorola 68331 25-MHz processor. The Khepera model that we used is a table-top robot, connected to a workstation through a wired serial link. This configuration allows an optional experimental configuration with everything at hand: the robot, the environment, and the host computer.

Different experimental world maps have been developed to study the performance of the three different Q-learning and the corresponding path-planning algorithms. The starting and the goal states S and G are marked in all the experimental maps. The snapshots of each map after the construction of the trajectory by the robot using distinctive colored lines for the three algorithms CQL, EQL, and IQL for all the experiments are given.

The first experiment is developed based on the world map shown in Fig. 13. The IQL and the EQL respectively take 179 and 1710 iterations to learn the said environment. The CQL algorithm with random selection requires 3012 iterations for convergence. After the learning phase is over, the path-planning algorithm is executed by keeping the Khepera at state no. 48, facing left, in the experimental world map and is allowed to traverse to the goal using the stored Q-table by all the three algorithms. The paths taken by the robot with the stored Q-table by the IQL, CQL, and EQL are shown in Fig. 13 by red, blue, and green lines, respectively.

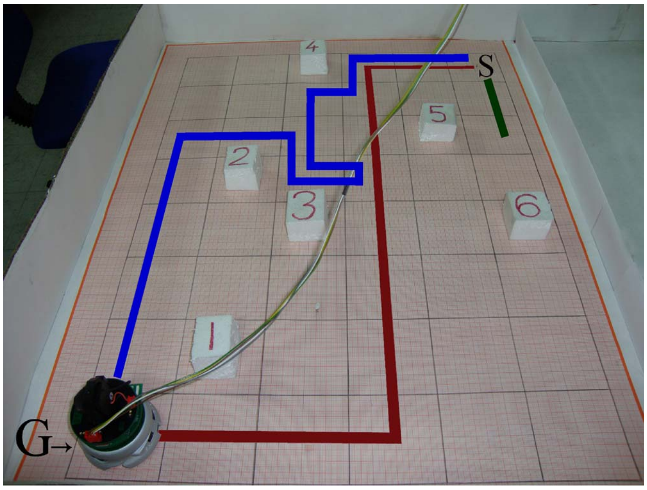

Fig. 14. World map with six obstacles added after the learning phase. Path planned by the robot: 🟥 Using the Q-table returned by the IQL. 🟦 Using the Q-table returned by the CQL. 🟩 Using the Q-table returned by the EQL (The robot fails to reach the goal. After reaching state no. 36, the next state is state no. 35, but an obstacle is present in that state. The robot will stop at state no. 36).

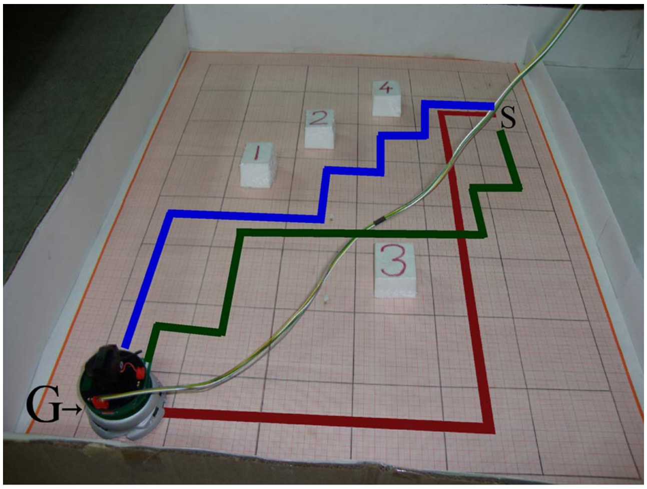

Fig. 15. World map with four obstacles. Path planned by the robot: 🟥 Using the Q-table returned by the IQL. 🟦 Using the Q-table returned by the CQL. 🟩 Using the Q-table returned by the EQL.

The second experiment considers Q-learning in Fig. 14 and path planning in the same world map after introducing six rectangular obstacles. The paths taken by the robot with the stored Q-table by IQL, CQL, and EQL are shown in Fig. 14 by distinctive colored lines. The robot with the stored Q-table by the EQL fails to reach the goal as indicated by the black line in Fig. 14. After reaching state no. 36, the best action stored in the Q-table is the RIGHT action. In order to perform this action, the robot turns right by 90° and checks for the presence of any obstacle in front of it with the help of sensors 2 and 3. The robot finds an obstacle in state no. 35, and there is no alternative action stored in the Q-table. Therefore, the robot with the stored Q-table obtained by the EQL fails to reach the goal and stops at state no. 36.

Experiment 3 is concerned with learning and planning in the world map shown in Fig. 15 containing four obstacles, numbered 1 to 4. All the three algorithms are used to learn the

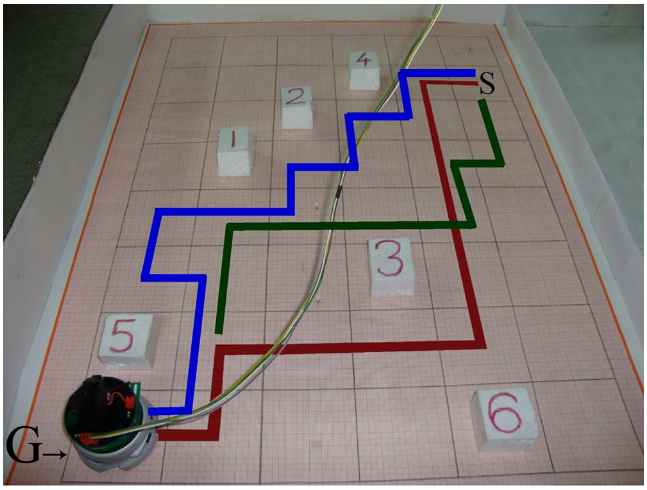

Fig. 16. World map with four obstacles. Path planned by the robot: 🟥 Using the Q-table returned by the IQL. 🟦 Using the Q-table returned by the CQL. 🟩 Using the Q-table returned by the EQL (When the robot reaches state no. 8, the best action at state no. 8 is to move left, but an obstacle is present at state no. 7. Therefore, the robot will stop at state no. 8).

movement steps from each grid in the map to its neighboring grid. The IQL takes 311 iterations to learn the said environment, while the EQL algorithm takes 1857 iterations for the same learning task. The CQL algorithm with random action selection takes 3060 iterations for convergence. After the completion of the learning phase, we kept the Khepera, facing left, in the same environment, and the planning phase is executed with all three algorithms by using the stored Q-table obtained by the respective learning algorithm. The red, blue, and green lines in Fig. 15 show the paths taken by the robot with the stored Q-table by IQL, CQL, and EQL, respectively.

In experiment 4, we train the robot in the world map given in Fig. 16 with four obstacles numbered 1 to 4. After the training is over, we add two obstacles numbered 5 and 6 in the map, and the planning cycle is executed in the modified world map given in Fig. 16. The path taken by the robot with the stored Q-table by IQL, CQL, and EQL are given by red, blue, and green lines, respectively. The robot with the stored Q-table by the EQL fails to reach the goal as indicated by the incomplete green line segment in Fig. 16. The justification of the failure is given hereinafter. After reaching state no. 8, the robot determines the best action at this state, which is to move left, but an obstacle is present at the next state. Therefore, the robot reports failure and stops in state no. 8.

Results of the experiments undertaken earlier are summarized in Table II. The table compares the relative performances of the IQL, EQL, and CQL in path planning. The metrics employed to compare the relative merits of the Q-learning algorithms in the planning phase are the following: 1) time taken to reach the goal; 2) the number of 90° turns involved in the path planning; and 3) the number of states traversed during the planning phase.

It is apparent from Table II that, for all the world maps shown in Figs. 13–16, the IQL outperforms the CQL and the EQL with respect to all the three metrics. The maximum noteworthy merit of the IQL over the others is the number of 90° turns, which is a bare minimum as evident from Table II.

TABLE II
COMPARISON OF TIME TAKEN BY THE ROBOT, NUMBER OF 90° TURNS REQUIRED BY THE ROBOT, AND NUMBER OF STATES TRAVERSED BY THE ROBOT

Fig. No	Time Taken in sec			No. of 90° turn			No. of states traversed		
	IQL	*EQL*	*CQL*	*IQL*	*EQL*	*CQL*	*IQL*	*EQL*	*CQL*
13	40.73	48.44	47.40	1	5	4	12	12	12
14	43.74	-	71.17	2	-	9	12	-	16
15	39.62	48.40	48.40	2	7	7	11	11	11
16	43.72	-	59.52	4	-	10	11	-	13

IQL	Improved Q-learning
EQL	Extended Q-learning
CQL	Classical Q-learning
-	Goal cannot be reached

VIII. CONCLUSION

This paper proposed an alternative algorithm for deterministic Q-learning, presuming that the background knowledge about the distance from the current state to both the next state and the goal state are available. The proposed algorithm updates the entries of the Q-table only once unlike the CQL, where the entries in the Q-table were updated many times until convergence was ensured. This results in a significant saving in the time complexity of the order of mn in comparison to the CQL, where n and m are the number of states and number of actions at each state, respectively.

Theorem 1 indicates the correctness in the steady-state values in the entries of the Q-table. Time-complexity analysis also reveals that the proposed algorithm saves a time complexity of the order of mn, when compared to the CQL.

Experiments simulated on different experimental mazes and on the Khepera platform confirm the better performance of the proposed algorithm in comparison to the CQL and the EQL algorithms. The Q-table updated by the IQL, when used for path-planning application, outperforms both the CQL and the EQL with respect to all the three metrics used in the experimental study. Most importantly, the 90° turnings required in the IQL are significantly reduced in comparison to the CQL and the EQL. Since the IQL outperforms CQL and EQL in all the three metrics as indicated in Tables I and II, it has a good potential in path-planning applications of mobile robots, particularly when obstacles are added in real time.

REFERENCES

[1] T. Dean, K. Basye, and J. Shewchuk, "Reinforcement learning for planning and control," in *Machine Learning Methods for Planning and Scheduling*, S. Minton, Ed. San Mateo, CA: Morgan Kaufmann, 1993.

[2] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[3] C. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, May 1992.

[4] A. Konar, *Computational Intelligence: Principles, Techniques and Applications*. New York: Springer-Verlag, 2005.

[5] L. Busoniu, R. Babushka, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL: CRC Press, 2010.

[6] J. Chakraborty, A. Konar, L. C. Jain, and U. Chakraborty, "Cooperative multi-robot path planning using differential evolution," *J. Intell. Fuzzy Syst.*, vol. 20, no. 1/2, pp. 13–27, Apr. 2009.

[7] M. Gerke and H. Hoyer, "Planning of optimal paths for autonomous agents moving in inhomogeneous environments," in *Proc. 8th Int. Conf. Adv. Robot.*, Jul. 1997, pp. 347–352.

[8] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 18–28, Apr. 1997.

[9] Z. Bien and J. Lee, "A minimum-time trajectory planning method for two robots," *IEEE Trans. Robot. Autom.*, vol. 8, no. 3, pp. 443–450, Jun. 1992.

[10] M. Moll and L. E. Kavraki, "Path planning for minimal energy curves of constant length," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2004, pp. 2826–2831.

[11] R. Regele and P. Levi, "Cooperative multi-robot path planning by Heuristic priority adjustment," in *Proc. IEEE/RSJ Int Conf Intell. Robots Syst.*, 2006, pp. 5954–5959.

[12] Y.-P. Hsu, W.-C. Jiang, and H.-Y. Lin, "A CMAC-Q-learning based Dyna agent," in *Proc. SICE Annu. Conf.*, Tokyo, Japan, 2008, pp. 2946–2950.

[13] Y. Zhou and M. J. Er, "A novel Q-learning approach with continuous states and actions," in *Proc. 16th IEEE Int. Conf. Control Appl.*, Singapore, Oct. 1–3, 2007, pp. 18–23.

[14] K. Cho, Y. Sung, and K. Um, "A production technique for a Q-table with an influence map for speeding up Q-learning," in *Proc. Int. Conf. Intell. Pervasive Comput.*, 2007, pp. 72–75.

[15] D. Pandey and P. Pandey, "Approximate Q-learning: An introduction," in *Proc. 2nd Int. Conf. Mach. Learn. Comput.*, 2010, pp. 317–320.

[16] S. S. Masoumzadeh, G. Taghizadeh, K. Meshgi, and S. Shiry, "A fuzzy Q-learning enhanced active queue management scheme," in *Proc. Int. Conf. Adapt. Intell. Syst.*, 2009, pp. 43–48.

[17] I. Goswami, P. K. Das, A. Konar, and R. Janarthanan, "Extended Q-learning algorithm for path-planning of a mobile robot," in *Proc. 8th Int. Conf. SEAL*, Dec. 2010, pp. 379–383.

[18] L. A. Jeni, Z. Istenes, P. Korondi, and H. Hashimoto, "Hierarchical reinforcement learning for robot navigation using the intelligent space concept," in *Proc. 11th Int. Conf. Intell. Eng. Syst.*, Budapest, Hungary, Jun. 29–Jul. 1, 2007, pp. 149–153.

[19] T. Martínez-Marín and R. Rodríguez, "Navigation of autonomous vehicles in unknown environments using reinforcement learning," in *Proc. IEEE Intell. Veh. Symp.*, Istanbul, Turkey, Jun. 13–15, 2007, pp. 872–876.

[20] W. Kwon, I. H. Suh, S. Lee, and Y.-J. Cho, "Fast reinforcement learning using stochastic shortest paths for a mobile robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, San Diego, CA, Oct. 29–Nov. 2, 2007, pp. 82–87.

[21] L. A. Jeni, Z. Istenes, P. Szemes, and H. Hashimoto, "Robot navigation framework based on reinforcement learning for intelligent space," in *Proc. Conf. Human Syst. Interact.*, Krakow, Poland, May 25–27, 2008, pp. 761–766.

[22] R. Lang, S. Kohlhauser, G. Zucker, and T. Deutsch, "Integrating internal performance measures into the decision making process of autonomous agents," in *Proc. Conf. Human Syst. Interact.*, Rzeszow, Poland, May 13–15, 2010, pp. 715–721.

[23] G. Tesauro, "Extending Q-learning to general adaptive. Multi-agent systems," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, vol. 16, pp. 871–878.

[24] P. Frazier and W. B. Powell, "The knowledge gradient policy for offline learning with independent normal rewards," in *Proc. IEEE Symp. ADPRL*, 2007, pp. 143–150.

[25] J.-J. Park, J.-H. Kim, and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," *Int. J. Control, Autom., Syst.*, vol. 5, no. 6, pp. 674–680, Dec. 2007.

[26] S. Lu, X. Liu, and S. Dai, "Incremental multistep Q-learning for adaptive traffic signal control based on delay minimization strategy," in *Proceedings of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China, Jun. 25–27, 2008, pp. 2854–2858.

[27] W. Chen, J. Guo, X. Li, and J. Wang, "Hybrid Q-learning algorithm about cooperation in MAS," in *Proc. CCDC*, 2009, pp. 3943–3947.

[28] E. Gomes and R. Kowalczyk, "Dynamic analysis of multiagent Q-learning with E-greedy exploration," in *Proc. 26th Int. Conf. Mach. Learn.*, 2009, vol. 382, pp. 369–376.

[29] Z. Chen and R. C. Qiu, "Q-learning based bidding algorithm for spectrum auction in cognitive radio," in *Proc. IEEE Southeastcon*, Mar. 2011, pp. 409–412.

[30] A. Galindo-Serrano and L. Giupponi, "Distributed Q-learning for aggregated interference control in cognitive radio networks," *IEEE Trans. Veh. Technol.*, vol. 59, no. 4, pp. 1823–1834, May 2010.

[31] O. Alsaleh, B. Hamdaoui, and A. Fern, "Q-learning for opportunistic spectrum access," in *Proc. 6th Int. Wireless Commun. Mobile Comput. Conf.*, 2010, pp. 220–224.

[32] C. Wu, K. Chowdhury, and M. D. Felice, "Spectrum management of cognitive radio using multi-agent reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2010, pp. 1705–1712.

[33] R. C. Qiu, Z. Chen, N. Guo, Y. Song, P. Zhang, H. Li, and L. Lai, "Towards a real-time cognitive radio network testbed: Architecture, hardware platform, and application to smart grid," in *Proc. 5th IEEE Workshop Netw. Technol. Softw.-Defined Radio White Space*, Jun. 2010, pp. 1–6.

[34] W. Yaping and Z. Zheng, "A method of reinforcement learning based automatic traffic signal control," in *Proc. 3rd Int. Conf. Meas. Technol. Mechatron. Autom.*, 2011, pp. 119–122.

[35] H. Okamura and T. Dohi, "Application of reinforcement learning to software rejuvenation," in *Proc. 10th Int. Symp. Autonom. Decentral. Syst.*, 2011, pp. 647–652.

[36] S. Mahadevan, "Average reward reinforcement learning: Foundations, algorithms, and empirical results," *Special Issue on Reinforcement Learning, Mach. Learn.*, vol. 22, no. 1–3, pp. 159–196, Jan.–Mar. 1996.

[37] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL: CRC Press, 2010.

[38] Y. Sakaguchi and M. Takano, "Reliability of internal prediction/estimation and its application. I. Adaptive action selection reflecting reliability of value function, neural networks," vol. 17, no. 7, pp. 935–952, Sep. 2004.

[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Boston, MA: MIT Press, 1998.

[40] [Online]. Available: http://www.computationalintelligence.net/main/main_page.html

Amit Konar (SM'10) received the B.E. degree from the Bengal Engineering and Science University (B.E. College), Howrah, India, in 1983, and the M.E. Tel E, M. Phil., and Ph.D. (Engineering) degrees from Jadavpur University, Calcutta, India, in 1985, 1988, and 1994, respectively.

In 2006, he was a Visiting Professor with the University of Missouri, St. Louis. He is currently a Professor with the Department of Electronics and Telecommunication Engineering, Jadavpur University, where he is the Founding Coordinator of the M.Tech. program on intelligent automation and robotics. He has supervised 15 Ph.D. dissertations. He has over 250 publications in international journal and conference proceedings. He is the author of eight books, including the two popular texts "Artificial Intelligence and Soft Computing" (CRC Press, 2000) and "Computational Intelligence: Principles, Techniques and Applications" (Springer, 2005). His research areas include the study of computational intelligence algorithms and their applications to the various domains of electrical engineering and computer science. Specifically, he worked on fuzzy sets and logic, neurocomputing, evolutionary algorithms, Dempster–Shafer theory, and Kalman filtering and applied the principles of computational intelligence in image understanding, very large scale integration design, mobile robotics, pattern recognition, brain–computer interfacing, and computational biology.

Dr. Konar serves as an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART-A and the IEEE TRANSACTIONS ON FUZZY SYSTEMS. He was the recipient of the All India Council for Technical Education-accredited 1997–2000 Career Award for Young Teachers for his significant contribution in teaching and research.

Indrani Goswami Chakraborty received the B.E. degree in electrical engineering, the M.E. degree (with specialization in control engineering) in electronics and communication engineering, and the Ph.D. degree in cognitive robotics from Jadavpur University, Calcutta, India, in 1993, 1988, and 2012, respectively, where she is currently pursuing her research and working toward another Ph.D. degree.

She was a Visiting Research Scientist in the Robotics Laboratory, Mie University, Nagoya, Japan. Her current research interest includes machine intelligence and robotics, industrial control, cognitive science, brain–computer interfacing, and evolutionary algorithms. She has published a number of interesting papers in top international journals and conference proceedings.

Sapam Jitu Singh received the B.E. degree in computer technology from Nagpur University, Maharashtra, India, in 2000, and the M.E. degree in electronics and telecommunication engineering (computer engineering specialization) from Jadavpur University, Calcutta, India, in 2011.

Since 2003, he has been working as a Lecturer with the Department of Computer Science and Engineering, Manipur Institute of Technology, Manipur, India. His research interests include evolutionary computing, digital image processing, and robotics.

Lakhmi C. Jain is currently a Professor of knowledge-based engineering and the Director/ Founder of the Knowledge-Based Intelligent Engineering Systems Centre, University of South Australia, Adelaide, Australia. His research interests focus on artificial intelligence paradigms and their applications in complex systems, art–science fusion, e-education, e-healthcare, robotics, unmanned air vehicles, and intelligent agents.

Prof. Jain is a Fellow of the Institution of Engineers Australia.

Atulya K. Nagar received the B.Sc.(Hons.), M.Sc., and M.Phil. (with distinction) from the MDS University of Ajmer, Rajasthan, India.

He holds the Foundation Chair, as Professor of Computer and Mathematical Sciences, at Liverpool Hope University, Liverpool, U.K., and is the Head of the Department of Mathematics and Computer Science. Prior to joining Liverpool Hope, he was with the Department of Mathematical Sciences and, later, with the Department of Systems Engineering at Brunel University, London, U.K. A mathematician by training, he possesses multidisciplinary expertise in natural computing, bioinformatics, operations research, and systems engineering. He has an extensive background and experience of working in universities in the U.K. and India. He has been an expert reviewer for the Biotechnology and Biological Sciences Research Council, grants peer-review committee for the Bioinformatics Panel, and serves on the Peer-Review College of the Arts and Humanities Research Council as a scientific expert member.

Prof. Nagar has coedited volumes on intelligent systems and applied mathematics. He is the Editor-in-Chief of the International Journal of Artificial Intelligence and Soft Computing and serves on editorial boards for a number of prestigious journals as well as on the International Programme Committee for several international conferences. He received a prestigious Commonwealth Fellowship for working toward his Doctorate in Applied Nonlinear Mathematics, which he earned from the University of York, U.K., in 1996.