COLLABORATIVE Q(\(\lambda\)) REINFORCEMENT LEARNING ALGORITHM - A PROMISING ROBOT LEARNING FRAMEWORK


*Department of Industrial Engineering and Management, Ben-Gurion University of the Negev
Be’er-Sheeva, 84105, Israel
{kartoun, helman, yael}@bgu.ac.il

**Institute for Medical Informatics, Washington Hospital Center
110 Irving St., Washington DC, NW, 20010, U.S.A.
{cfiedy}@ncemi.org

ABSTRACT
This paper presents the design and implementation of a new reinforcement learning (RL) based algorithm. The proposed algorithm, \(CQ(\lambda)\) (collaborative \(Q(\lambda)\)) allows several learning agents to acquire knowledge from each other. Acquiring knowledge learnt by an agent via collaboration with another agent enables acceleration of the entire learning system; therefore, learning can be utilized more efficiently. By developing collaborative learning algorithms, a learning task solution can be achieved significantly faster if performed by a single agent only, namely the number of learning episodes to solve a task is reduced. The proposed algorithm proved to accelerate learning in navigation robotic problem. The \(CQ(\lambda)\) algorithm was applied to autonomous mobile robot navigation where several robot agents serve as learning processes. Robots learned to navigate an 11 x 11 world contains obstacles and boundaries choosing the optimum path to reach a target. Simulated experiments based on 50 learning episodes showed an average improvement of 17.02% while measuring the number of learning steps required reaching definite optimality and an average improvement of 32.98% for convergence to near optimality by using two robots compared with the \(Q(\lambda)\) algorithm [1, 2].

KEY WORDS
Robot simulation, reinforcement learning, and navigation

1. Introduction

Reinforcement learning (RL) used in this work is learning through direct experimentation [3, 4]. It does not assume the existence of a teacher that provides training examples. Instead, in RL experience is the only teacher. The learner receives signals (reinforcements) from the process by getting an indication about how well it is performing the required task. These signals are usually associated to some dramatic condition - e.g., accomplishment of a subtask (reward) or complete failure (punishment), and the learner’s goal is to optimize its behavior based on some performance measure (maximization of a cost function). The learning agent learns the associations between observed states and chosen actions that lead to rewards or punishments, i.e., it learns how to assign credit to past actions and states by correctly estimating costs associated to these events [5].

Navigation task can be broken down into three parts [6]: (i) localization, the process of figuring out where the robot is; (ii) mapping, the process whereby the robot builds a model of its environment, and (iii) planning, the process of figuring out how the robot can get to other places. Robots are inherently uncertain about the state of their environments. Uncertainty arises from sensor limitations, noise and the fact that real-world environment is unpredictable. Learning to navigate in realistic environments requires novel algorithms for identifying important events and find efficient action policies.

In [7] navigation learning of a miniature mobile robot is described. The robot equipped with vision capabilities learns to navigate a maze using several RL-based algorithms. [8] demonstrate a two mode \(Q\)-learning on a humanoid robot at a 17 x 17 maze for improving \(Q\)-learning performance. A RL algorithm for accelerating acquisition of new skills by real mobile robot is described in [9]. The algorithm speeds up \(Q\)-learning by applying memory-based sweeping [10] and was tested within an image-based visual servoing framework on an ActivMedia PeopleBot mobile robot for a docking task. A solution for robotic docking based on neural and reinforcement is presented in [11]. The solution was achieved partially by training of a value function unit and four motor units via RL. [12] describe a collaborative process enabling a robotic learner to acquire concepts and skills from human examples. During teaching the robot
The robot executes the tasks and by incorporating feedback its hypothesis space is converged. With Q-learning approach, the robot learns a button pushing task.

Although Q-learning and Q(λ) were used in many fields of robotics [e.g., 13, 10, 14, 15, 16, 17], the issue of acceleration of learning is still significant. It includes the acceleration of learning toward finding an optimal or close to optimal solution. In order to improve learning, we suggest the new CQ(λ) algorithm. The CQ(λ) algorithm proved to accelerate learning in a system composed of several similar learning processes.

The paper is organized as follows. In Section two, reinforcement learning theory is described. That includes description of the Q, Q(λ) and the new CQ(λ) algorithms. Section three describes the learning system. Section four demonstrates simulation experiments and results of applying the CQ(λ) algorithm on a navigation problem. Concluding remarks follow in Section five.

2. Reinforcement Learning

The basic assumption on the study of RL is that any state \( s_{t+1} \) made by the agent must be a function only of its last state and action: \( s_{t+1} = f(s_t, a_t) \) where \( s_t \in S \) is the state at time-step \( t \) and \( a_t \in A \) is the action taken. Naturally, if the agent can faithfully observe the states of the process which by definition summarize all the relevant information about the process dynamics at a certain instant of time then its observations are Markovian. On the other hand, if the observations made by the agent are not sufficient to summarize all the information about the process, a non-Markovian condition takes place:

\[
s_{t+1} = f(s_t, s_{t-1}, s_{t-2}, ..., a_t, a_{t-1}, a_{t-2}, ...) [5].
\]

It is usually more adequate to express the dynamic of the process through a collection of conditional transition probabilities \( P(s_{t+1} \in S | s_t \in S, a_t \in A) \). Of particular interest is the discounted infinite horizon formulation of the MDP (Markov Decision Process) problem. Given a finite set of possible actions \( a \in A \), a finite set of process states \( s \in S \), a stationary discrete-time stochastic process, modeled by transition probabilities \( P(s_{t+1} | s_t, a_t) \) and a finite set of bounded reinforcements \( r(s_t, a_t) \in R \), the agent tries to find out a stationary policy of actions \( a_t^* = \pi^*(s_t) \) which maximizes the expected cost function (Eq. 1):

\[
V^\pi(s_t) = \lim_{{M \to \infty}} E_{{s'_t \in S}} \left[ \sum_{{t=0}}^{\infty} \gamma^t r(s_t, \pi(s_t)) \right],
\]

for every state \( s_t \). \( \pi^* \) indicates the dependency on the followed action policy, via the transition probabilities \( P(s_{t+1} | s_t, \pi(s_t)) \). The discount factor \( 0 \leq \gamma < 1 \) forces recent reinforcements to be more important than remote ones. The optimal cost function is presented in Eq. 2 where it is possible that there will be more than a single optimal policy \( \pi^* \) [18].

\[
V^*(s_0) = \lim_{{M \to \infty}} E_{{s'_t \in S}} \left[ \sum_{{t=0}}^{\infty} \gamma^t r(s_t, \pi^*(s_t)) \right].
\]

2.1. Q-Learning

The RL algorithm Q-learning [1], is modified here. In Q-learning the system estimates the optimal action-value function directly and then uses it to derive a control policy using the local greedy strategy. The advantage of Q-learning is that the update rule is model free as it is a rule that just relates Q values to other Q values. It does not require a mapping from actions to states and it can calculate the Q values directly from the elementary rewards observed. Q is the system’s estimate of the optimal action-value function [19]. It is based on the action value measurement \( Q(s_t, a_t) \), defined in Eq. 3:

\[
Q(s_t, a_t) = E[r(s_t, a_t) + \gamma V^*(s_{t+1})] = r(s_t, a_t) + \gamma \sum_{{s_{t+1} \in S}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}),
\]

which represents the expected discounted cost for taking action \( a_t \) when visiting state \( s_t \) and following an optimal policy thereafter. From this definition and as a consequence of the Bellman’s optimality principle [20], Eq. 4 is derived:

\[
Q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{{s_{t+1} \in S}} P(s_{t+1} | s_t, a_t) \max_{{a_t}} Q(s_{t+1}, a_t).
\]

These characteristics (max operator inside the expectation term and policy independence) allow an iterative process for calculating an optimal action policy via action values which is the essence of Q-learning. The first step of the algorithm is to initialize the system’s action-value function, Q. Since no prior knowledge is available, the initial values can be arbitrary (e.g., uniformly zero). Next, the system’s initial control policy, \( P \), is established. This is achieved by assigning to \( P \) the action that locally maximizes the action-value. At time-step \( t \), the agent
visits state \( s_i \in S \) and selects an action \( a_i \in A \), receives from the process the reinforcement \( r(s_i, a_i) \in R \) and observes the next state \( s_{i+1} \). Then it updates the action value \( Q(s_i, a_i) \) according to Eq. 5 which describes a Q-learning one step:

\[
Q_{t+1}(s_i, a_i) = (1 - \alpha)Q_t(s_i, a_i) + \alpha [ r(s_i, a_i) + \gamma \hat{V}_t(s_{i+1}) ],
\]

where \( \hat{V}_t(s_{i+1}) = \max_{a_i \in A}[Q_t(s_{i+1}, a_i)] \) is the current estimate of the optimal expected cost \( V^*(s_{i+1}) \) and \( \alpha \) is the learning rate which controls how much weight is given to the reward just experienced, as opposed to the old \( Q \) estimate. The process repeats until stopping criterion is met (e.g., robot reached target). \( \alpha = 1 \) gives full weight to new experiences. As \( \alpha \) decreases, the \( Q \)-value is built up based on all experiences, and new unusual experience does not disturb the established \( Q \)-value much. The greedy action \( \hat{V}_t(s_{i+1}) = \max_{a_i \in A}[Q_t(s_{i+1}, a_i)] \) is the best the agent performs when in state \( s_i \), but for the initial stages of the learning process it uses randomized actions that encourages exploration of the state-space. Under some reasonable conditions [21] this is guaranteed to converge to the optimal \( Q \)-function [19].

### 2.2. CQ(\( \lambda \))-Learning

\( Q(\lambda) \) [1, 2] is a generalization of \( Q \)-learning that uses eligibility traces, \( e(s_i, a_i) \): the one-step \( Q \)-learning is a particular case with \( \lambda = 0 \). The \( Q \)-learning algorithm learns quite slowly because only one time-step is traced for each action. To boost convergence of learning, the multi-step tracing mechanism, the eligibility trace, is used, in which the \( Q \) values of a sequence of actions can be updated simultaneously according to the respective lengths of the eligibility traces [13]. An outline of the multi-step \( Q(\lambda) \) learning algorithm [1, 2], which is based on the tableau version in [22], is shown in Fig. 1.

The new \( CQ(\lambda) \) (Fig. 2) algorithm objective is to accelerate learning in a system composed of several similar learning processes. It is based on that a state-action value of an agent or learning process is updated according to the maximal value within all other learning processes state-action values exist in a learning system. Alternatively if only two learning agents are involved such as a robot and human, state-action values known to the human are acquired by the robot learning function.

![Fig. 1. Q(\( \lambda \)) learning algorithm](image1)

![Fig. 2. CQ(\( \lambda \))-learning algorithm](image2)
the value of $Q(s, a)$. This is known as a “softmax” action selection. A common method is to use a Gibbs or Boltzmann distribution, where the probability of choosing action $a_t$ in state $s_t$ is proportional to $e^{\beta Q(s_t, a_t)}$, i.e., in state $s_t$ the agent chooses action $a_t$ with probability

$$P(a_t | s_t) = \frac{e^{\beta Q(s_t, a_t)}}{\sum_{a_i} e^{\beta Q(s_t, a_i)}},$$

(6)

where $\beta_i$ is a positive parameter which specifies how randomly values should be chosen. When $\beta_i$ is low, the actions are chosen about the same amount each other. As $\beta_i$ increases, the highest valued actions are more likely to be chosen, and in the limit $\beta_i \rightarrow \infty$ the best action is always chosen [23].

In implementing the $CQ(\lambda)$ we adapt the attitude described in [4] for one agent and apply it for the collaborative agent attitude: learning rate of each process $i \in \{1, 2, ..., N\}$, $\alpha_i$, is set relatively high and is reduced adaptively over time. This is done independently for each state-action pair; number of times each state-action pair has been previously updated, $c_{i(s,a)}$, is calculated. The effective learning rate, $\alpha_{i(s,a)}$, is then determined from the initial rate by

$$\alpha_{i(s,a)} = \frac{\alpha_{i(s,a)}^0}{c_{i(s,a)} + 1}.$$

(7)

The principle is that the more and more knowledge about a certain state-action pair is gained, the less it is should be modified in response to any particular experience. Convergence was proved by [21].

3. Learning System

Robots' states include robot locations in a $11 \times 11$ two dimensional world (Fig. 3). The state $s_t$ is defined by:

$$s_{(i,j)} = \{i, j\} \text{ where } i \in \{1, 2, ..., 11\} \text{ and } j \in \{1, 2, ..., 11\}.$$

Actions can be taken at each state are; traveling north, west, south and east. An action, $a_k$, is noted as $a_{(k)}$, where $k \in \{1, 2, 3, 4\}$. Rewards defined as $r_{(l)}$, where $l \in \{-1, 0, +1\}$. If the robot reaches target, the reward is +1. If it passes through an undesirable area such as an obstacle / boundary, the reward is -1. Otherwise, the reward is 0. Learning episode is a description of one session of reaching the target. Performance measures include: (i) $N_{i_{near-optimal}}$ - convergence to near optimality - mean of the last $N_i$ learning step values, and (ii) $N_{i_{optimal}}$ - convergence to optimality - number of learning steps required to achieve the shortest path and repeat it infinite number of times where $i \in \{1, 2, ..., N\}$ and $N$ is the number of learning processes.

4. Experimental Results

The simulated system was evaluated using two experimental model sets and significance levels of mean steps to converge values were examined. In both models robot agents learn the world simultaneously and share knowledge. The first model contains two robot agents where the first learns according to $CQ(\lambda)$ and the second learns according to the $Q(\lambda)$ algorithms. The second model contains three robot agents where the first learns according to $CQ(\lambda)$ and the second and third learn according to the $Q(\lambda)$ algorithms.

At the first experimental model contains two robot agents. The first robotic agent noted as $i = 1$ learns according to the $CQ(\lambda)$ algorithm, i.e., gathers knowledge from the other robot whereas the second agent noted as $i = 2$ learns according to the $Q(\lambda)$ algorithm and does not gain knowledge from the first robot. The following parameters were set: $N = 2, N_t = 10, \alpha_1 = \alpha_2 = 0.95$ (initial values),
\( \gamma_1 = \gamma_2 = 0.99 \), and \( \lambda_1 = \lambda_2 = 0.5 \) (Fig. 2). At the second experimental model contains three robot agents, the first robot noted as \( i = 1 \) learns according to the \( CQ(\lambda) \) algorithm, i.e., gathers knowledge from the other two robots whereas the second and third robots noted as \( i = 2 \) and \( i = 3 \) learn according to the \( Q(\lambda) \) algorithm and do not gain knowledge from the first robot. The model was set with the following parameters: \( N = 3 \), \( N_i = 10 \), \( \alpha_1 = \alpha_2 = \alpha_3 = 0.95 \) (initial values), \( \gamma_1 = \gamma_2 = \gamma_3 = 0.99 \), and \( \lambda_1 = \lambda_2 = \lambda_3 = 0.5 \) (Fig. 2).

Experiments based on 50 simulation runs were conducted for both models. An example for state-action values of one simulation run is given in Fig. 4. In both models number of learning episodes was set to 100, i.e., the algorithm stops after an agent navigates from starting point to target 100 times. For evaluating system performance, a specific state with coordinates \((3, 2)\) was chosen. For this state, the optimal route length traveling to the target at coordinates \((8, 11)\) is 14 steps (Fig. 3). Based on the results shown in Table 1, ten hypotheses were evaluated (Table 2).

Null hypotheses \( H_{10} \) and \( H_{20} \) were rejected with P-values equal to 2.58 \( \times \) 10\(^{-5}\) and 0 respectively; namely; the mean coefficients of learning agents are not equal. Null hypothesis \( H_{30} \) was not rejected with P-value equals to 0.643 which results an equal means of the two learning agents. Null hypotheses \( H_{40} \) and \( H_{50} \) were rejected with P-values equals to 4.52 \( \times \) 10\(^{-6}\) and 6.59 \( \times \) 10\(^{-7}\) respectively concluding a difference between the means of the learning agents. The null hypothesis \( H_{60} \) was not rejected with P-value equals to 0.824; namely; the means of these two learning agents are equal. Null hypotheses \( H_{70} \) and \( H_{80} \) were rejected with P-values equals to 1.33 \( \times \) 10\(^{-15}\) and 0 respectively; namely; there is a difference between the means of the learning agents. Fig. 5 shows an example of one of the 50 simulation runs using two robots to converge (Table 1).

![Fig. 4. 11 x 11 world state-action value map after 100 learning episodes](image)

For evaluating whether adding a third learning agent improves learning, hypotheses nine and ten were tested (Table 2). Null hypotheses \( H_{90} \) and \( H_{100} \) were not rejected with P-value equals to 0.81 and 0.436 respectively; namely; the means are equal. This means that an additional learning agent does not improve system learning significantly.

5. Discussion and Conclusions

A new type of a collaborative reinforcement learning algorithm has been developed in traditional navigation task. We demonstrated in simulation an acceleration of learning and the convergence to an optimum using...
collaboration between several learning agents. We presented the design and implementation via simulation of the $CQ(\lambda)$ algorithm applied on autonomous mobile robots for navigation in an 11 x 11 two dimensional world contains obstacles and boundaries choosing the optimum path to reach a target. 50 simulation runs showed an average improvement of 17.02% while measuring the number of learning steps required reaching definite optimality and an average improvement of 32.98% for convergence to near optimality by using two robots compared with the $Q(\lambda)$ algorithm. Significant statistical difference was indicated for both convergence to optimality and convergence to near optimality while comparing two robots; the first uses $CQ(\lambda)$ and the second uses $Q(\lambda)$ . While using three robots; the first uses the $CQ(\lambda)$ and the second and third use $Q(\lambda)$, we found that there is no statistical significant differences in both convergence to optimality and convergence to near optimality while comparing the $Q(\lambda)$ -based robots. We found statistical significant differences in both convergence to optimality and convergence to near optimality while comparing the $CQ(\lambda)$ -based robot to the other two. In addition we found that there is no statistical significant differences in both convergence to optimality and convergence to near optimality using $CQ(\lambda)$ -based robot learns in a two robots environment or an $CQ(\lambda)$ -based robot learns in three robots environment, i.e., we conclude that adding another robot in addition to two collaborative robots does not achieve significant improvement. We demonstrated: (i) superiority of the $CQ(\lambda)$ algorithm over the $Q(\lambda)$ and (ii) proved statistically that three learning processes have no significant advantage over two learning processes, thereby when $CQ(\lambda)$ will be tested on a real robot, only one additional learning process (e.g., another robot or a human) will be applied.

Basically there is always a gap between simulated and real robot tests; the sensors and actuators are never ideal and are usually very difficult to model accurately and the operation environment is usually dynamic, etc. Future work includes verification of the $CQ(\lambda)$ algorithm with real robots for: (i) learning the efficient lifting and shaking policies for emptying the contents of suspicious bags using a fixed arm robot [27, 28, 29, 30], and (ii) learning a navigation task using an Evolution Robotics ER-I robot [31] when a human serves as an additional learning process.

6. Acknowledgements

This project was partially supported by the Paul Ivanier Center for Robotics Research and Production Management, Ben-Gurion University of the Negev.

References:


### Table 1. Simulation results

<table>
<thead>
<tr>
<th>Learning strategy</th>
<th>Experimental model set I</th>
<th>Experimental model set II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean / standard deviation of steps to converge to near optimality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robot1</td>
<td>C(Q(λ))</td>
<td>36.8 / 7.2</td>
</tr>
<tr>
<td>Robot2</td>
<td>Q(λ)</td>
<td>68.4 / 10.1</td>
</tr>
<tr>
<td>Robot3</td>
<td>Q(λ)</td>
<td>55.2 / 6.5</td>
</tr>
<tr>
<td>Robot4</td>
<td>Q(λ)</td>
<td>68.9 / 15.6</td>
</tr>
<tr>
<td>Robot5</td>
<td>Q(λ)</td>
<td>69.4 / 11.2</td>
</tr>
</tbody>
</table>

### Table 2. Evaluation hypotheses

<table>
<thead>
<tr>
<th>Evaluation within experimental model set I learning agents</th>
<th>H₀₀: There is no difference between convergence to optimality of the two learning agents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>H₀₁: There is a significant difference between the two learning agents.</td>
<td>H₀₂: There is no difference between convergence to near optimality of the two learning agents.</td>
</tr>
<tr>
<td>H₀₃: There is a significant difference between the two learning agents.</td>
<td>H₀₄: There is no difference between convergence to optimality of C(Q(λ)) learning (Robot2) and Q(λ) learning (Robot3).</td>
</tr>
<tr>
<td></td>
<td>H₀₅: There is a significant difference between the two learning agents.</td>
</tr>
<tr>
<td>Evaluation within experimental model set II learning agents</td>
<td>H₁₀: There is no difference between convergence to optimality of Q(λ) learning (Robot2) and Q(λ) learning (Robot3).</td>
</tr>
<tr>
<td>H₁₁: There is a significant difference between the two learning agents.</td>
<td>H₁₂: There is no difference between convergence to near optimality of Q(λ) learning (Robot2) and Q(λ) learning (Robot3).</td>
</tr>
<tr>
<td>H₁₃: There is a significant difference between the two learning agents.</td>
<td>H₁₄: There is no difference between convergence to near optimality of C(Q(λ)) learning (Robot1) and Q(λ) learning (Robot2).</td>
</tr>
<tr>
<td>H₁₅: There is a significant difference between the two learning agents.</td>
<td>H₁₆: There is no difference between convergence to near optimality of C(Q(λ)) learning (Robot1) and Q(λ) learning (Robot2).</td>
</tr>
<tr>
<td>H₁₇: There is a significant difference between the two learning agents.</td>
<td>H₁₈: There is no difference between convergence to near optimality of C(Q(λ)) learning (Robot1) and Q(λ) learning (Robot2).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Evaluation between experimental sets I and II</th>
<th>H₂₀: There is no difference between convergence to optimality of using two or three C(Q(λ)) robots.</th>
</tr>
</thead>
<tbody>
<tr>
<td>H₂₁: There is a significant difference between the two learning models.</td>
<td>H₂₂: There is a significant difference between the two learning models.</td>
</tr>
<tr>
<td>H₂₃: There is no difference between convergence to optimality of using two or three C(Q(λ)) robots.</td>
<td>H₂₄: There is a significant difference between the two learning models.</td>
</tr>
</tbody>
</table>