Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 3, No. 1, April 2017.

BAT Q-LEARNING ALGORITHM

Bilal H. Abed-alguni

(Received: 30-Nov.-2016, Revised: 01-Feb.-2017, Accepted: 23-Feb.-2017)

ABSTRACT

Cooperative Q-learning approach allows multiple learners to learn independently and then share their Q-values among each other using a Q-value sharing strategy. A main problem with this approach is that the solutions of the learners may not converge to optimality, because the optimal Q-values may not be found. Another problem is that some cooperative algorithms perform very well with single-task problems, but quite poorly with multi-task problems. This paper proposes a new cooperative Q-learning algorithm called the Bat Q-learning algorithm (BQ-learning) that implements a Q-value sharing strategy based on the Bat algorithm. The Bat algorithm is a powerful optimization algorithm that increases the possibility of finding the optimal Q-values by balancing between the exploration and exploitation of actions by tuning the parameters of the algorithm. The BQ-learning algorithm was tested using two problems: the shortest path problem (single-task problem) and the taxi problem (multi-task problem). The experimental results suggest that BQ-learning performs better than single-agent Q-learning and some well-known cooperative Q-learning algorithms.

KEYWORDS

Q-learning, Bat algorithm, Optimization, Cooperative reinforcement learning.

1. INTRODUCTION

Q-learning is a well known reinforcement learning (RL) algorithm that allows machines and software agents to develop an ideal behavior within a specific environment based on trial and error [1]-[3]. A Q-learning agent learns how to behave by trying actions to determine how to maximize some reward. This is usually accomplished using temporal difference learning to find mapping from state-action pairs into quality values (Q-values). A Q-value of a state-action pair (s, a) represents the expected utility of taking action a in state s and following a fixed policy thereafter. The Q-values are normally calculated using a utility function known as a Q-function. These values are usually stored in a data structure known as a Q-table.

Cooperation among several reinforcement learners in the same multi-agent environment provides an opportunity for the learners to cooperatively solve a learning problem. Such an approach to RL, which is called cooperative RL, is increasingly used by research labs around the world to solve real world problems, such as robot control and autonomous navigation [4], [5]. This is because cooperative reinforcement learners can learn and converge faster than independent reinforcement learners via sharing of information (e.g., Q-values, Episodes, Policies) [3], [6]-[8]. One such example is cooperative Q-learning, in which several learners share their Q-values among each other in order to accelerate their convergence to optimal solutions [9], [10]. Cooperative Q-learning is normally broken into two stages. The first stage is known as the independent learning stage, in which each reinforcement learner individually applies Q-learning to enhance its own solution. In the second stage, the learning by interaction stage, the learners share their Q-values based on a sharing strategy. A Q-value sharing strategy defines how the independent learners can share their Q-values among each other to obtain new Q-tables. This strategy can only be applied when the agents have Q-tables with a similar structure.

B. Abed-alguni is with the Computer Science Department, Yarmouk University, Irbid, Jordan.
E-mail: Bilal.h@yu.edu.jo.

Current cooperative Q-learning algorithms, such as AVE-Q, BEST-Q, PSO-Q [6], [11]-[15] and WSS [7], [16]-[19], may not find the optimal Q-values for different reasons (Section 3). As a result, the policies of the learners might not converge to optimality. In addition, some cooperative Q-learning algorithms perform well with single-task problems, but very poorly with multi-task problems [9]. This issue causes uncertainty about the benefit of choosing one cooperative algorithm over the other cooperative algorithms.

The bat algorithm (BA) is a metaheuristic method that can be used to solve optimization problems by simulating the echolocation behavior of bats [20]. An advantage of BA is that it tries to balance between exploration and exploitation of actions by using tuning techniques that control its parameters (frequencies, pulse emission rates and loudness of the potential solutions) [20]. Consequently, the possibility of finding the optimal solutions increases. Therefore, in order to solve the problems of current cooperative Q-learning algorithms, this paper presents a new cooperative Q-learning algorithm called the Bat Q-learning algorithm (BQ-learning) that is based on the BA algorithm. BQ-learning is distinguished from the other cooperative algorithms by the use of a Q-value sharing strategy based on the BA algorithm. This paper argues that the proposed BQ-learning algorithm increases the possibility of finding the optimal Q-values for different types of learning problems.

The remainder of the paper is organized as follows: Section 2 presents background information, Section 3 discusses related work, Section 4 discusses the BQ-learning algorithm, Section 5 discusses simulation results using the shortest path problem and the taxi domain problem and Section 6 presents the conclusion and future work of this paper.

2. BACKGROUND INFORMATION

This section briefly summarizes some of the underlying concepts of Q-learning and Bat algorithms.

2.1 Q-learning

The problem model of Q-learning is commonly represented as a Markov Decision Process (MDP) [1]. An MDP comprises a set of states $S = \{s_0, s_1, ..., s_n\}$, a set of actions $A = \{a_0, a_1, ..., a_m\}$, a reward function $R : S \times A \rightarrow \mathbb{R}$ and a transition model $T : S \times A \times S \rightarrow [0,1]$. As specified by the transition model, all the transition probabilities are deterministic, meaning that they can only equal 1 or 0. For example, a transition probability $T(s_x, a_z, s_y) = 1$ means that transitioning from state s_x to state s_y upon executing action a_z is possible. On the other hand, a transition probability $T(s_x, a_z, s_y) = 0$ indicates that the transition is invalid. The immediate expected reward for executing this transition is the deterministic reward $R(s_x, a_z)$ [3]. It is important to note that the implementation of Q-learning to stochastic MDPs is beyond the scope of this paper.

A learner is normally required to apply Q-learning to an MDP for a number of learning episodes in order to learn which action is optimal for each state. A learning episode is the time the agent takes to reach the goal state starting from an initial selected state. Reaching the goal state requires the learner to apply a simple value iteration procedure during each learning episode. This procedure starts when the learner uses its selection policy to select an action a from the set of possible actions A of current state s. The learner then receives a reward $R(s,a)$ and bserves a new state s' of the environment. Subsequently, the agent uses these information to update its Q-table using the following Q-function:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,[R(s,a) + \gamma\, max_{a' \in A} Q(s',a')] \qquad (1)$$

where $s \in S$, $a \in A$, $\alpha \in [0,1]$ is the learning rate and $\gamma \in [0,1]$ is the discount factor. Upon successful convergence to a solution, the output of Q-learning is the optimal Q-function from which an optimal policy $\pi^* : S \rightarrow A$ (i.e., mapping from states to actions that maximizes the total discounted reward ($R = r_0 + \gamma^1 r_1 + \ldots + \gamma^n r_n$)) can be derived using a greedy selection method.

2.2 Bat Algorithm

Microbats are small bats that usually eat insects. An amazing feature of this species is that they rely on a special type of sonar called echolocation to locate their prey. Microbats make loud sound pulses as they fly. When these pulses hit an object, they produce echoes that return to the ears of the bats. The time required for the sound waves to return back to the microbat is used to calculate the distance of an object.

The Bat algorithm (BA) is a metaheuristic method that is inspired from the echolocation behavior of microbats [20]. This algorithm combines the advantages of existing metaheuristic algorithms, such as particle swarm optimization (PSO) and intensive local search in one algorithm. The research works of Yang and Gandomi [21] and Yang [20] suggest that BA performs better than many existing metaheuristic algorithms, such as PSO, intensive local search, harmony search and genetic algorithm.

The following simplifications of the main characteristics of the echolocation process were followed in order to simulate BA as a problem solver [22]:

- Microbats know the difference between prey and other objects and use echolocation to calculate the distance of their prey.
- Each bat i flies randomly at position x_i with velocity v_i, frequency f_i, varying wavelength λ and loudness A to hunt a prey.
- Loudness varies in the $[A_{min}, A_0]$ interval.
- Each bat i can adjust the frequency f_i and the pulse rate $r_i \in [0,1]$ of its emitted pulse.
- Frequencies of the bats are in the range $[f_{min}, f_{max}]$. These frequencies correspond to wave lengths in the range $[\lambda_{min}, \lambda_{max}]$ that can be calculated as follows:

$$\lambda = \frac{v}{f}; \text{where } v = 340 \ m/s \text{ which is the speed of sound in the air.} \quad (2)$$

Based on the above equation, either λ or f can be used in the BA algorithm, because the relationship between these variables is constant ($v = \lambda f$). The choice between λ and f depends on the type of the problem of interest.

At the beginning of BA (Figure 1), each bat is assigned a random frequency in the range $[f_{min}, f_{max}]$. This range is normally chosen based on the size and complexity of the implemented problem. There are several rules that control the movement of a virtual bat.

The following rules show how a virtual bat i changes its position x_i (solution) and velocity v_i at instant t:

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \quad (3)$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_*) f_i, \quad (4)$$

1: Objective function $fun(x)$, $x = (x_1,..., x_d)^T$
2: Initialize the bat population $x_i (i = 1, 2, ..., n)$ and v_i
3: Define pulse frequency f_i at x_i
4: Initialize pulse rates r_i to positive values around zero and loudness A_i to positive values around 1.
5: **while** *t< Max number of iterations* **do**
6: Generate new solutions by adjusting frequency and updating velocities and
7: locations/solutions [Equations (3) to (5)].
8: **if** *rand* > r_i **then**
9: Select a solution among the best solutions
10: Generate a local solution around the selected best solution
11: **endif**
12: Generate a new solution by flying randomly
13: Calculate the average loudness A^* of all solutions
14: **if** *rand* < A^* *and fun*(x_i) < *fun*(x_*) **Then**
15: Accept the new solutions
16: Increase r_i and reduce A_i
17: **endif**
18: Rank the bats and find the current best x_*
19: $t = t + 1$
20: **endwhile**
21: Postprocess results and visualization

Figure 1. The Bat algorithm (BA) [20].

$$x_i^t = x_i^{t-1} + v_i^t, \tag{5}$$

where $\beta \in [0,1]$ is a random parameter extracted from a uniform distribution and x_* is the current best position among the positions of all bats.

After calculating x_*, a local solution can be generated randomly for each bat based on the following equation:

$$x_{new} = x_{old} + \eta\, A^* \tag{6}$$

where $\eta \in [-1,1]$ is a tuning random parameter and A^* is the average loudness of all bats at instant t.

The update equations of the velocities, positions and frequencies of the bats are similar to the update equations of the velocities and positions of the particles in PSO (Section 3). Actually, BA can be considered as a combination of PSO and intensive local search that aims to balance between the exploration and exploitation of solutions.

In the nature, when a microbat finds a prey, it usually decreases the loudness and increases the pulse emission of its sound. This aspect is simplified in the BA algorithm by assuming that $A_{min}=0$ and $A0=1$. The assumption that $A_{min}=0$ indicates that a bat has located a prey and temporarily has stopped emitting any sound. In the beginning of the simulation process of BA, positive random values around zero are generated and assigned to the pulse emission of each bat, while positive random values around 1 are generated and assigned to the loudness of each bat.

At each iteration of the BA algorithm, a local search for a new best solution around one of the best solutions x_* (line 8) is triggered when the pulse rate is less than a randomly generated number $rand \in [0,1]$. Then, each time x* is improved (line 14) the pulse rate r_i is increased and the loudness A_i is decreased as follows:

$$A_i^{t+1} = \sigma\, A_i^t \tag{7}$$

$$r_i^{t+1} = r_i[1 - e^{-\delta t}] \tag{8}$$

where σ and δ are constant parameters that can be determined experimentally, however, as a general rule $0<\sigma<1$ and $\delta>0$ to guarantee that the loudness will decrease and the pulse rate will increase as new best solutions are discovered.

A new solution x_* is accepted if it satisfies two conditions. First, the estimation *fun(x_*)* of the new x_* must be better than the estimation of a randomly selected bat's solution. Second, the value of *rand* should be less than the average loudness of all the solutions.

The purpose of setting the loudness to a value near to one and the pulse rate to a value near to zero is to encourage the exploration of new solutions around the current best solutions. This is because a pulse rate near to zero is expected with a high probability to be less than the randomly generated number *rand* $\in [0,1]$ (line 8). Consequently, there is a high probability that a new solution would be generated around one of the best solutions (lines 8 to 11). As the values of pulse rates are increased each time a better solution is accepted (line 14), the probability of generating a new solution around one of the best solutions decreases (line 8).

3. RELATED WORK

This section provides an overview of well known cooperative Q-learning algorithms with special focus on the second learning stage of these algorithms.

Iima and Kuroe [11]-[13] proposed three cooperative Q-learning algorithms (BEST-Q, AVE-Q and PSO-Q) that allow multiple learners to share their Q-values after each round of independent learning. Each one of these algorithms evaluates its Q-values during the independent learning stage using an evaluation method that approximates the rewards [6], [13], [14]. This method evaluates each state-action pair (s,a) by calculating the sum of its discounted rewards $E(s,a)$ used to update its Q-value during an episode (independent learning stage). Discounting the reward is important to increase the weight of rewards while approaching the end of the episode. This is because the Q-values are in continuous change during the episode. At the end of the independent learning stage, each learner i calculates $E_i(s,a)$ for each $(s,a) \in S \times A$ as follows:

$$E_i(s,a) = \sum_{i=1}^{n} \gamma^{n-i}\, R_i(s,a) \tag{9}$$

where n denotes the number of times the state-action pair (s,a) has been updated by agent i during the episode, $R_i(s,a)$ is the reward received for performing action a at state s and γ is the discount parameter. The parameter γ is the same discount factor used in Equation 1. This parameter is used in Equation 9 to balance between the rewards received in the beginning of the episode with rewards received in the end of it.

In BEST-Q, the superior Q-values are extracted from the Q-tables of all of the learners, then copied to each Q-table of each agent. According to this description, an agent i updates $Q_i(s,a)$ for all $(s,a) \in S \times A$ as follows:

$$Q_i(s,a) \leftarrow Q^{best}(s,a). \quad (10)$$

In the above equation, $Q^{best}(s,a)$ of state-action pair (s,a) is the Q-value with the highest $E(s,a)$ for all agents. The main disadvantage of BEST-Q is that it might not find the optimal Q-values, because the Q-tables of all of the learners become the same after each update. As a result, the diversity of the Q-values is affected negatively [11].

AVE-Q is a modification of BEST-Q that retains the diversity of each agent's Q-values after the learning by interaction stage. In this algorithm, the Q-values of learner i are updated by averaging each Q-value in the learners' Q-table and its corresponding best Q-value for all $(s,a) \in S \times A$ as follows:

$$Q_i(s,a) \leftarrow \frac{Q^{best}(s,a) + Q_i(s,a)}{2}. \quad (11)$$

Actually, AVE-Q moves at the interaction stage into the middle of the agent's Q-values and their corresponding best values without investigating the quality of the agent's Q-values. As a consequence, AVE-Q may produce an incorrect policy, because it does not remove the bad Q-values at the interaction stage [3].

The Particle Swarm Optimization (PSO) algorithm is a powerful metaheuristic method that attempts to iteratively optimize a solution with respect to a particular measure [23]. An optimization problem can be solved with PSO by moving the candidate solutions (particles) in the search space based on their positions and velocities. The movement of a particle is controlled by the particle's local best position and directed in the direction of the global best positions in the search-space. The global best positions are the best positions found by all of the particles after each iteration of the algorithm.

PSO-Q uses PSO at its second learning stage as a Q-value sharing method. In this method, the particles are the Q-values and the qualitative measurer is the Q-function. The Q-table of each learner is updated based on the velocities and positions of the Q-values as follows [12]:

$$V_i(s,a) \leftarrow W\, V_i(s,a) + C_1 R_1 [P_i(s,a) - Q_i(s,a)] + C_2 R_2 [G(s,a) - Q_i(s,a)], \quad (12)$$

$$Q_i(s,a) \leftarrow Q_i(s,a) + V_i(s,a), \quad (13)$$

where V_i is the velocity of learner i for state-action pair (s,a), W, C_1 and C_2 are weight parameters and R_1 and $R_2 \in [0,1]$ are random numbers. In the above equation, the best Q-value found by agent i for (s,a) is denoted as $P_i(s,a)$ and the best Q-value found by all of the agents for (s,a) is denoted as $G(s,a)$. Normally, the value of $G(s,a)$ is estimated using Equation 10.

Two issues should be taken into consideration when implementing PSO-Q to a specific problem. First, determining suitable values for the parameters of PSO-Q usually requires multiple simulations to insure that PSO-Q will perform well. Second, there is no guarantee that PSO-Q will search outside the surroundings of the best Q-value for each possible combination of states and actions for all agents.

Ahmadabadi and Asadpour [18] proposed a cooperative Q-learning algorithm called Weighted Strategy Sharing (WSS). In this algorithm, each learner learns from its peers by following a two-step learning process. First, each learner assigns expertness values to the Q-tables of the other learners according to their relative expertness. Second, each learner updates its own Q-table by calculating the weighted average of the Q-values of the learners' Q-tables as follows:

$$Q_i(s,a) \leftarrow \sum_{j=1}^{n}(W_{ij}\ Q_j(s,a)) \tag{14}$$

where W_{ij} is the expertness value assigned by learner i to learner j's expertness.

An expertness value can be evaluated using one of many expertness measures that have similar outcomes [18]. One such measure is the Normal measure (Nrm) which calculates the expertness of a learner (xr) by finding the sum of rewards that the learner has obtained during the previous independent learning stage:

$$xr_i^{Nrm} = \sum_{t=1}^{now} r_i(t)\ ; \tag{15}$$

where $r_i(t)$ is the reward that learner i obtains at instant t.

Based on the output of the above formula, learner i can assign a weight to the knowledge of learner j by taking into account the expertness of all learners as follows:

$$W_{ij} \leftarrow \frac{xr_i}{\sum_{k=1}^{n} xr_k}\ ; \tag{16}$$

where n is the number of learners and xr_k is the expertness of learner k for $k = 1, ..., n$.

A problem with WSS is that it might not converge to optimality when the shared Q-values are so extreme, because these values will deform the average Q-value [9].

Abed-alguni et al. [9] suggested a new cooperative Q-learning algorithm called average aggregation Q-learning which combines WSS, AVE-Q, BEST-Q and PSO-Q into one algorithm in order to reduce the instability in the performance of these algorithms for different problems. In this algorithm, each agent improves its Q-values by averaging the Q-values that resulted after implementing WSS, BEST-Q, AVE-Q and PSO-Q algorithms. Respectively, each agent i calculates $Q_i(s,a)$ for each $(s,a) \in S \times A$ as follows:

$$Q_i(s,a) \leftarrow \frac{Q^{BEST-Q}(s,a) + Q^{AVE-Q}(s,a) + Q^{WSS}(s,a) + Q^{PSO-Q}(s,a)}{4}\ ; \tag{17}$$

where i is the learner's identification number and the denominator is the number of the combined algorithms.

Although average aggregation Q-learning solves the variability in performance for four famous cooperative Q-learning algorithms, it requires heavy computations to do so, because it mainly depends on the results of the other cooperative Q-learning algorithms.

In conclusion, there is no guarantee that the algorithms discussed in this section will converge to optimal solutions. Moreover, none of these algorithms has a stable performance when implemented to various learning problems [9]. The next section will present the BQ-learning algorithm that attempts to solve these problems.

4. BQ-LEARNING

The BQ-learning algorithm comprises two repetitive sequential learning stages.

- First Learning Stage: each learner tries independently to enhance its policy by applying Q-learning. Then, the Q-values of all the agents are evaluated by the evaluation method described in Section 3 - Equation 9.

- Second Learning Stage: the Q-values of all of the learners are updated through sharing of Q-values among the learners according to the evaluation results of the Q-values and the bat Q-value sharing strategy .

4.1 First Stage of BQ-learning

Figure 2 shows the BQ-learning algorithm. In the beginning of BQ-learning, the number of learners n and the total number of episodes of BQ-learning p should be specified. Also, the number of learning episodes m_i that each learner i performs during the first learning stage of BQ-learning should be specified. In addition, the Q-values and Q-value evaluations of each learner are initialized to zero (lines 8 to 10). That is, $Q_i(s,a) = 0$ and $E_i(s,a) = 0$ for all $(s,a) \in S \times A$ of each learner i.

```
 1: Q_i : Q -table of learner i .
 2: Q_i(s,a) : Q-value for state-action pair (s,a) of learner i .
 3: E_i(s,a) : evaluated value for Q_i(s,a) .
 4: n : number of learners.
 5: m_i : number of learning episodes performed by learner i during the first
        learning stage of BQ-learning.
 6: p : total number of episodes of BQ-learning.
 7: Begin
 8: for i = 1 to n do
 9:   Initialize Q_i(s,a) and E_i(s,a) for all (s,a) ∈ S × A .
10: end for
11: Set counter = 0 .
12: while counter < p do
13:   for i = 1 to n do
14:     Allow learner i to apply Q-learning as described in Section 2.1 for m_i
        episodes.
15:     Find E_i(s,a) for all (s,a) ∈ S × A based on Equation (9).
16:   end for
17:   Update Q_i(s,a) for all (s,a) ∈ S × A for all of the learners by interaction
        between the learners based on the bat Q-value sharing strategy described in
        Figure 3.
18:   counter = counter + n * m .
19:  end while
20: End
```

Figure 2. The BQ-learning algorithm.

Lines 13 to 16 in Figure 2 represent the first learning stage of BQ-learning, where each learner i applies Q-learning for m_i episodes and then calculates $E_i(s,a)$ for all $(s,a) \in S \times A$ as described in Section 2.1. Allowing each learner to learn for the same number of episodes ($m_0 = m_1 = ... = m_{n-1} = m_n$) indicates that all of the n learners have equal levels of knowledge at the end of the first learning stage. On the other hand, allowing each learner to learn for a different number of episodes means that the learners have different levels of knowledge at the end of the first learning stage.

4.2 Second Stage of BQ-learning

1: Q_i : Q -table of learner i .
2: $Q_i(s,a)$: Q-value for state-action pair (s,a) for learner i .
3: $Q_*(s,a)$: best Q-value for state-action pair (s,a) for all agents.
4: $F_i(s,a)$: frequency for state-action pair (s,a) for learner i .
5: $V_i(s,a)$: difference of $Q_i(s,a)$ before and after its update.
6: $r_i(s,a)$: pulse rate for state-action pair (s,a) for learner i .
7: $A_i(s,a)$: loudness for state-action pair (s,a) for learner i .
8: $E_i(s,a)$: evaluated value for state-action pair (s,a) for learner i .
9: $E_*(s,a)$: evaluated value for $Q_*(s,a)$.
10: $A^*(s,a)$: average loudness of all $A_i(s,a)$.
11: **Begin**
12: Set the objective function as the evaluation function of Q-values (Equation (9)).
13: **for** $i=1$ to n **do**
14: Initialize $F_i(s,a)$, $r_i(s,a)$, $A_i(s,a)$ and $V_i(s,a)$ for all $(s,a)\in S\times A$.
15: **endfor**
16: Find $Q_*(s,a)$ by applying the update function of BEST-Q algorithm (Equation (10)).
17: **while** $t <$ *Max number of iterations* **do**
18: **for** $i=1$ to n **do**
19: Update $V_i(s,a)$, $F_i(s,a)$ and $Q_i(s,a)$ for all $(s,a)\in S\times A$. [Equations 18 to 20].
20: **endfor**
21: Generate a random number ($rand\in[0,1]$).
22: **if** ($rand > r_*(s,a)$) **then**
23: Allow learner i^* to apply Q-learning for few times starting from state s of $Q_*(s,a)$ using Equation 21.
24: **endif**
25: Randomly select $Q_i(s,a)$.
26: **if** ($rand < A^*(s,a)$ **and** $E_i(s,a) < E_*(s,a)$) **then**
27: Accept the new Q-values.
28: Increase $r_i(s,a)$ and reduce $A_i(s,a)$ [Equations 22 and 23] .
29: **endif**
30: Find $Q_*(s,a)$ by applying the update function of BEST-Q algorithm (Equation (10)).
31: $t=t+1$
32: **endwhile**
33: **End**

Figure 3. Bat Q-value sharing strategy.

Line 17 in Figure 2 represents the second learning stage of BQ-learning that is described in details in Figure 3. It is important to keep in mind that the second learning stage of BQ-learning is what really distinguishes it from the other cooperative Q-learning algorithms described in Section 3.

Figure 3 shows the flow of the proposed Q-value sharing strategy that is based on the Bat algorithm. In Figure 3, the Q-values represent the locations of the bats (line 2), the velocity of a Q-value *V(s,a)* is the rate at which it changes (line 5) and the objective function is the evaluation

function *E(s,a)* of the Q-values (line 12). Line 14 in Figure 3 shows that the frequency, loudness and pulse rate for all $(s,a) \in S \times A$ of each learner i are initialized to zero. Then, in line 16, the best Q-value of each $(s,a) \in S \times A$ is calculated.

Line 19 of this algorithm shows that the Q-values and their frequencies and velocities are updated iteratively. The new Q-value $Q_i(s,a)$, velocity $V_i(s,a)$ and frequency $F_i(s,a)$ of learner i are given by:

$$F_i(s,a) = F_{min} + (F_{max} - F_{min})\beta, \tag{18}$$

$$V_i(s,a) = V_i(s,a) + (Q_i(s,a) - Q_*(s,a))F_i(s,a), \tag{19}$$

$$Q_i(s,a) = Q_i(s,a) + V_i(s,a), \tag{20}$$

where $\beta \in [0,1]$ is a random tuning parameter and $Q_*(s,a)$ is the current best Q-value among all n Q-values for the state-action pair (s,a).

A local Q-value can be generated around $Q_*(s,a)$ for each learner by allowing one of the learners to enhance $Q_*(s,a)$ by applying Equation 21 to $Q_*(s,a)$ for few times:

$$Q_*(s,a) \leftarrow (1-\alpha)\, Q_*(s,a) + \alpha\, [R(s,a) + A^* * max_{a \in' A} Q_*(s',a')]; \tag{21}$$

where $\alpha \in [0,1]$ is the same learning rate used in Equation 1 and A^* is the average loudness of all Q-values at iteration t. In the above equation, A^* is used to control the influence of future rewards instead of γ in Equation 1.

At each iteration of the algorithm, a local search for a new best Q-value (line 22) around the current best Q-value $Q_*(s,a)$ for each (s,a) is triggered when the pulse rate $r_*(s,a)$ is less than a randomly generated number $rand \in [0,1]$.

The new Q-value of $Q_*(s,a)$ is accepted if it satisfies two conditions. First, the estimation $E_*(s,a)$ of the new $Q_*(s,a)$ must be better than the estimation of a randomly selected Q-value for the same (s,a). Second, the value of *rand* should be less than the average loudness of (s,a) of all the learners. Fulfilling these conditions also implies that the pulse rate $r_i(s,a)$ should be increased and the loudness $A_i(s,a)$ should be decreased as follows:

$$A_i(s,a) = \sigma\, A_i(s,a), \tag{22}$$

$$r_i(s,a) = r_i(s,a)[1 - e^{-\gamma t}]; \tag{23}$$

where σ and δ are constant parameters. As a general rule, $0 < \sigma < 1$ to decrease the loudness and $\delta > 0$ to increase the pulse rate each time the Q-values are improved.

Assigning a low pulse rate $r_i(s,a)$ for each (s,a) in the beginning of the optimization process (line 17) and then increasing it (line 28) is an essential factor for the success of the algorithm. This is because it reduces the rate of local search around $Q_*(s,a)$ as BQ-learning is approaching the best Q-value.

The local search for the best Q-values can be performed simultaneously by multiple agents in BQ-learning as well as in other optimization-based cooperative Q-learning algorithms, such as PSO-Q and average aggregation Q-learning. BQ-learning is expected to perform better than the cooperative Q-learning algorithms discussed in Section 3, because it attempts to balance between the exploration and exploitation of the nominated best Q-values for sharing using tuning techniques that control its parameters (frequencies, pulse emission rates and loudness of

the potential solutions). Neither BEST-Q nor AVE-Q attempts to search around the best Q-values before sharing them. Consequently, BEST-Q might not find the optimal Q-values [11], while AVE-Q may produce an incorrect policy [11]-[12].

5. EXPERIMENTS

In this section, the performance of BQ-learning was compared with the performance of single-agent Q-learning, AVE-Q, BEST-Q, PSO-Q, WSS and average-aggregation Q-learning (Section 3) using two problems: the shortest path problem [12] and the taxi problem [24]. These problems have been widely used in the literature to evaluate the performance of cooperative Q-learning algorithms [12]-[13], [24]-[26].

5.1 Test Problems

RL can be applied to two types of learning problems [24]. First, single-task problems (e.g., shortest path problem), in which the learner is required to learn a single task. Second, multi-task problems (e.g., taxi domain problem), in which the learner is required to learn multiple tasks.

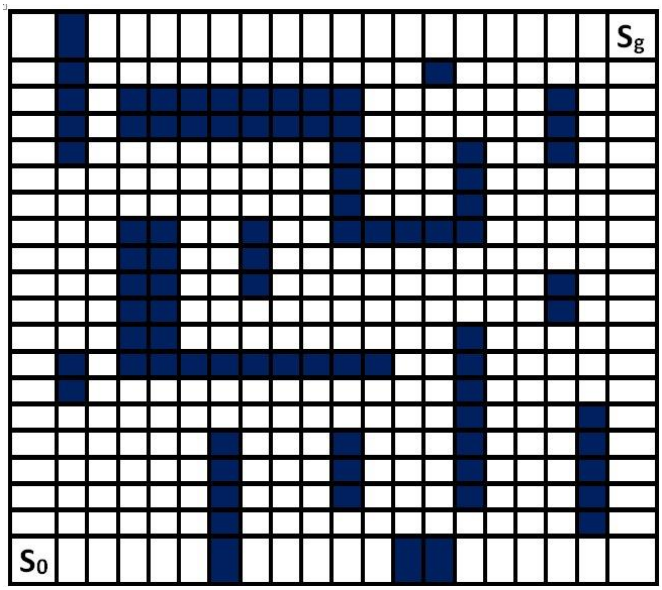

Figure 4. An example of shortest path problem on a grid size of 20×20.

The shortest path problem is a single-task problem that has been used in many research studies to evaluate the efficiency of cooperative Q-learning algorithms [11]-[13]. In this problem, an agent is required to learn one task which is finding the shortest path from one cell to another in a grid, such that the number of visited cells is minimized. The grid in this problem is usually represented as a two-dimensional array that is indexed by two subscripts, one for the row and one for the column. In the shortest path problem, the target cell is usually specified prior to learning and the start cell is randomly selected before the beginning of each learning episode. The learner can move during each episode in four directions (up, down, right and left) as long there are no obstacles or barriers obstructing its way. Figure 4 shows an example of shortest path problem on a grid size of 20×20. Filled squares represent obstacles that the agent cannot pass, s_0 is the start cell and s_g is the target cell.

The taxi domain problem is an episodic multi-task problem that has been used in many research studies to evaluate the performance of hierarchical Q-learning algorithms [24]-[26]. In each episode, a taxi agent in a grid world of size 5×5 is required to perform multiple tasks: finding a customer, picking up the customer, driving the customer to a destination location and dropping

down the customer in the destination location. The taxi agent can accomplish these goals by choosing actions from a set of six actions: move one cell (left, right, top or bottom), pickup action and drop off action. If any of these actions that leads the taxi agent to a barrier or a wall cell, the location of taxi agent remains unchanged. In the grid, there are four source and destination locations. Figure 5 shows an example of taxi domain problem. In the figure, a taxi is located on a 5×5 grid. There are four pre-determined locations in the grid, marked as Red (R), Blue (B), Green (G) and Yellow (Y). In the beginning of the simulation process, one of these locations is selected as a pick-up point and another location is selected as a drop-off point.

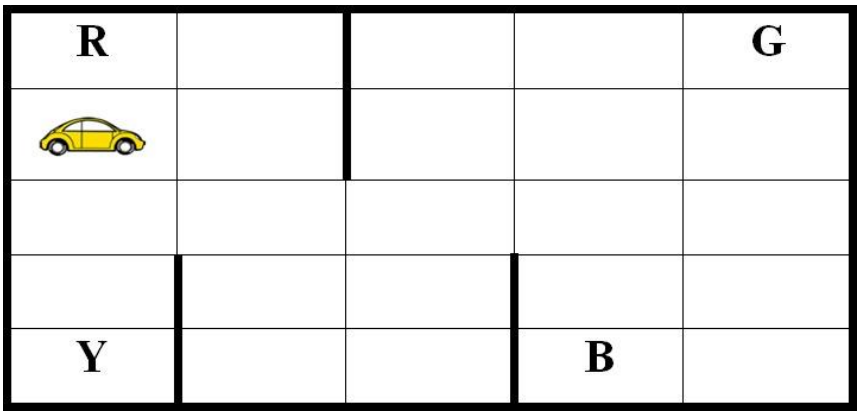

Figure 5. Taxi domain problem.

5.2 Setup

The shortest path problem in Figure 4 was modeled as an MDP as follows:

- The cells in the 20×20 grid represent the states of the MDP:

$$S = (grid\,[0][0],\ grid\,[0][1],\ ...,\ grid\,[19][19])\ .$$

- The target cell is specified prior to learning and the start cell is randomly selected before the beginning of each learning episode.
- There are four primitive actions in the shortest path problem:

$$A = (\text{move up, move down, move left, move right})\ .$$

- The reward model for the learners is defined as follows:

$$R(s,a) = \begin{cases} +10.0 & \text{if it reached the target cell} \\ 0 & \text{otherwise} \end{cases}$$

- The transition model for the learners is:

$$T(s,a,s') = \begin{cases} 1 & \text{if } s' \text{ is next to } s \text{ in the direction of } a \text{ and is} \\ & \text{not a barrier or wall} \\ 0 & \text{otherwise} \end{cases}$$

The taxi domain problem in Figure 5 was modeled as an MDP as follows:

- The cells in the 5×5 grid represent the states of the MDP:

$$S = (grid\,[0][0],\ grid\,[0][1],\ ...,\ grid\,[4][4])\ .$$

- The location of the taxi is specified prior to learning.

- The location of the passenger (source) and the dropping point of the passenger (destination) are randomly chosen in the beginning of each learning episode.
- There are six primitive actions in the taxi domain problem:

$$A = (\text{move up, move down, move left, move right, pick up, put down}).$$

- The reward model for the learners is defined as follows:

$$R(s,a) = \begin{cases} +10.0 & \text{If the passenger was delivered to its selected drop off point} \\ -10 & \text{illegal pickup or putdown} \\ 0 & \text{otherwise} \end{cases}$$

- The transition model for the learners is:

$$T(s,a,s') = \begin{cases} 1 & \text{if } s' \text{ is next to } s \text{ in the direction of } a \text{ and is not a barrier or wall} \\ 0 & \text{otherwise} \end{cases}$$

The experiments were implemented using two models of knowledge [19]. In the first model, the learners were assumed to have equal levels of knowledge. This was simulated by allowing the learners to learn for the same number of episodes before sharing of their Q-values. In the second model, the learners were assumed to have different levels of knowledge, which was achieved by allowing each learner to learn for a different number of episodes each time it is learning independently. For example, a learner that has learned for 25 episodes has more practical knowledge than a learner that has learned only for 10 episodes.

The action selection policy was the ε-soft policy, in which a random action is uniformly selected with probability ε and the action with the highest expected reward is chosen the rest of the time [22].

The learning parameters for the experiments were set as follows:

- In Q-learning, the learning rate α was tuned dynamically, so that low Q-values have larger learning rates than high Q-values as recommended by Ray and Oates [27]. The discount factor $\gamma = 1$ [28].
- In all the cooperative Q-learning algorithms, the learning rate $\alpha = 0.01$ and the discount factor $\gamma = 0.9$. These values ensure that each cooperative learner learns adequately and make, the best use of its current knowledge at each learning episode as recommended by Abed-alguni *et al.* [9].
- In each episode, a learner starts learning from a randomly selected state and finishes learning when a goal state is reached. Otherwise, the learner finishes learning after 5,000 moves without meeting its goal.
- In order to ensure an adequate exploration/exploitation ratio, the probability of selecting a random action was $\varepsilon = 0.05$ in the ε-soft selection policy.
- The Nrm measure was selected as the expertness measure of WSS. This measure has a similar performance to the performance of all other tested expertness measures.
- As in Abed-alguni et al. [9], the weight parameters in PSO-Q were $W = 0$, $C1 = C2 = 1$.
- In BQ-learning, the frequency, the loudness and the pulse rate were in the range [0,1] for each solution. The discount parameter of the frequencies β=*0.5*. Initially, the loudness A was set to 1 and the pulse rate r was set to 0 for each Q-value.

Three agents are involved in the experiments. The total number of learning episodes is 2,000 for the shortest path problem, while the total number of learning episodes for the taxi problem is

12,000 episodes. Each algorithm was executed 100 times in order to provide meaningful statistical analysis of the experiments.

In the experiments, an algorithm is considered to have converged to a good policy when the average number of moves in its policy enhances by less than one move over 100 successive episodes.

5.3 Experimental Results

5.3.1 Shortest Path Problem

Figure 6 shows the average number of moves per 10 episodes to find the shortest path to the goal state in a 20×20 grid. The second learning stage of the cooperative Q-learning algorithms takes place after each 10 episodes of individual learning. We can see from the figure that BQ-learning converges after 420 episodes to a solution. On the other hand, single agent Q-learning, AVE-Q, WSS, average aggregation Q-learning and PSO-Q converge after around 520 episodes to solutions, while BEST-Q requires 60 additional episodes to converge to a solution. These results suggest that the performance of BQ-learning is better than those of the other algorithms in single-task problems when the agents have similar levels of knowledge before sharing.

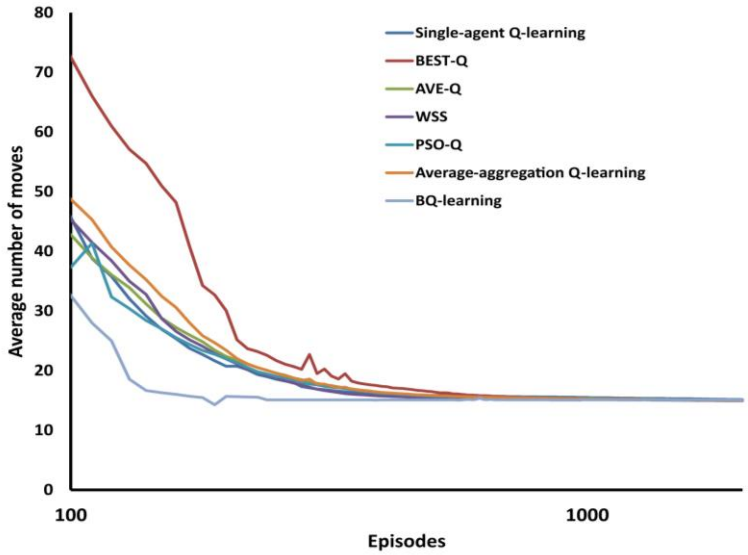

Figure 6. Experiment 1: Average number of moves per 10 episodes in a shortest path problem of a grid size of 20×20. Each curve is the average of 100 runs. Sharing of Q-values takes place after each ten episodes of individual learning.

Figure 7 shows the average number of moves per 10 episodes to find the shortest path to the goal state in a 20×20 grid. Respectively, in Figure 7, the first, the second and the third agents learn for 10, 5 and 1 episodes before sharing of their Q-values among each other. In this experiment, BQ-learning requires 300 episodes of learning to converge to a solution, which is only 14.9% of the number of episodes required for single-agent Q-learning (2020), 54% of BEST-Q (550), 53.6% of WSS (560), 62.5% of PSO-Q (480), 61.2% of AVE-Q (490) and 29.4% of average aggregation Q-learning (1,020). These results mean that BQ-learning outperforms the other algorithms in single-task problems when the agents have different levels of knowledge before sharing.

5.3.2 Taxi Problem

Figures 8 and 9 show the average number of steps per 10 episodes to deliver a passenger in a

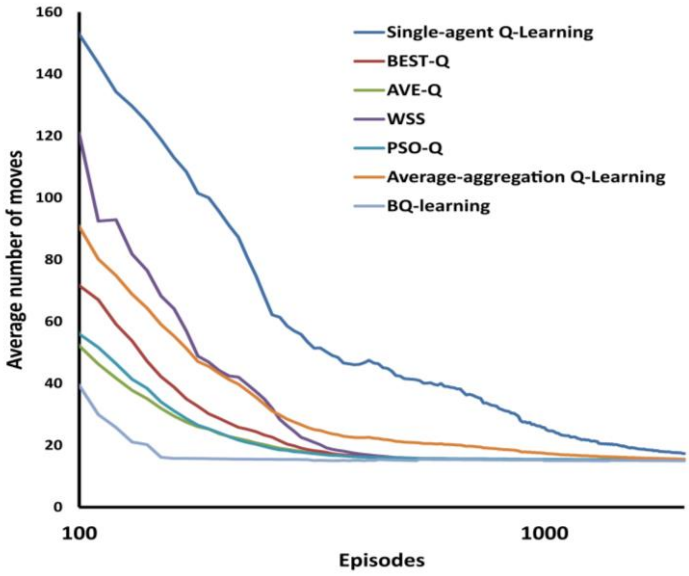

Figure 7. Experiment 2: Average number of moves per 10 episodes in a shortest path problem of a grid size of 20×20. Each curve is the average of 100 runs. One, five and ten episodes of learning occur before implementing a Q-value sharing strategy.

5×5 grid. Sharing of Q-values occurs in Figure 8 after each 10 episodes of independent learning, while in Figure 9, the 1st learner, the 2nd learner and the 3rd learner respectively learn for 1, 5 and 10 episodes before sharing of their Q-values.

Figure 8 shows that BQ-learning requires 7,180 episodes to converge to a solution, followed by PSO-Q that requires 7,560 episodes (5% more episodes than BQ-learning) to converge to a solution. On the other hand, the other algorithms failed to converge to a solution at the end of the simulation process. These results suggest that BQ-learning converges to a solution faster than the other algorithms in multi-task problems when the agents have similar experiences.

Figure 9 shows that all of the cooperative Q-learning algorithms failed to converge to a solution except BQ-learning and AVE-Q. As expected, BQ-learning has the fastest convergence speed among all algorithms. From Figure 9, we can also see that WSS has the worst performance among all the algorithms including single-agent Q-learning. These results indicate that BQ-learning outperforms the other algorithms in multi-task problems when the agents have different levels of experience.

5.4 Performance Analysis

Two statistical measures were used in Table 1 to compare the performance of the tested algorithms over 100 runs. The results are in the format: average number of iterations $\pm$ standard deviation of iterations. The last row of the table shows that BQ-learning requires less number of iterations to converge to a solution. In addition, the standard deviations of the number of iterations of BQ-learning are the lowest among all the standard deviations of the other algorithms that converge to a solution. This means that the performance of BQ-learning is more stable than the performance of the other tested algorithms.

Figures 10 and 11 show how the performance of three instances of BQ-learning is affected as the number of learning episodes of the agents is varied: (1-5-10), (15-30-45) and (25-50-100) learning episodes before sharing. The results in Figure 10 show that the convergence points of all instances of BQ-learning in the shortest path problem are not far from each other: BQ-learning (1-5-10) converges after 303 episodes, BQ-learning (15-30-45) converges after 333 episodes and BQ-learning (25-50-100) converges after 342 episodes.

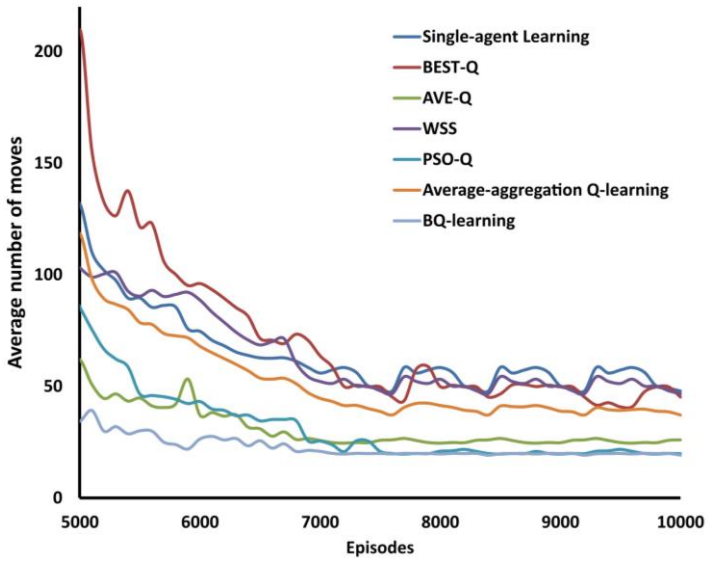

Figure 8. Experiment 3: Average number of moves per 10 episodes in a taxi problem of a grid size of 5×5. Each curve is the average of 100 runs. Sharing of Q-values takes place after each ten episodes of individual learning.

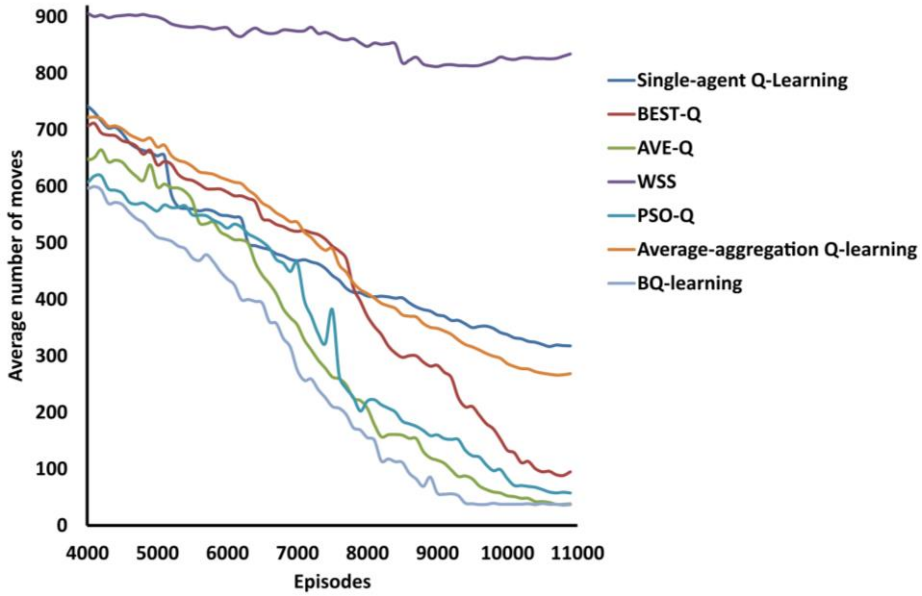

Figure 9. Experiment 4: Average number of moves per 10 episodes in a taxi problem of a grid size of 5×5. Each curve is the average of 100 runs. One, five and ten episodes of learning occur before implementing a Q-value sharing strategy.

In Figure 11 (taxi domain), BQ-learning (1-5-10) converges after 9,136 episodes, BQ-learning (15-30-45) converges after 9,233 episodes and BQ-learning (25-50-100) converges after 9,417 episodes. To sum up, the results in both figures indicate that the convergence speed of BQ-learning is not highly sensitive to the number of episodes that each agent learns before sharing of Q-values.

The overall results of the experiments suggest that BQ-learning performs better than conventional Q-learning and the other cooperative Q-learning algorithms, regardless of the

levels of experience of the agents (similar experiences vs. different experiences) and the types of the learning problems (single-task vs. multiple-task problems).

Table 1. Average and standard deviation of number of iterations over 100 runs. The star symbol * indicates that the algorithm did not converge to a solution at the end of the simulation process.

Algorithm	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Single-agent Q-learning	520 ± 36	$2{,}000 \pm 0^{*}$	$12{,}000 \pm 0^{*}$	$12{,}000 \pm 0^{*}$
BEST-Q	583 ± 44	550 ± 66	$12{,}000 \pm 0^{*}$	$12{,}000 \pm 0^{*}$
AVE-Q	521 ± 31	480 ± 35	$12{,}000 \pm 0^{*}$	$11{,}003 \pm 356$
WSS	524 ± 32	560 ± 44	$12{,}000 \pm 0^{*}$	$12{,}000 \pm 0^{*}$
PSO-Q	523 ± 34	480 ± 61	$7{,}560 \pm 701$	$12{,}000 \pm 0^{*}$
Average aggregation	521 ± 31	$1{,}020 \pm 52$	$12{,}000 \pm 0^{*}$	$12{,}000 \pm 0^{*}$
BQ-learning	413 ± 17	300 ± 32	$7{,}180 \pm 425$	$9{,}136 \pm 256$

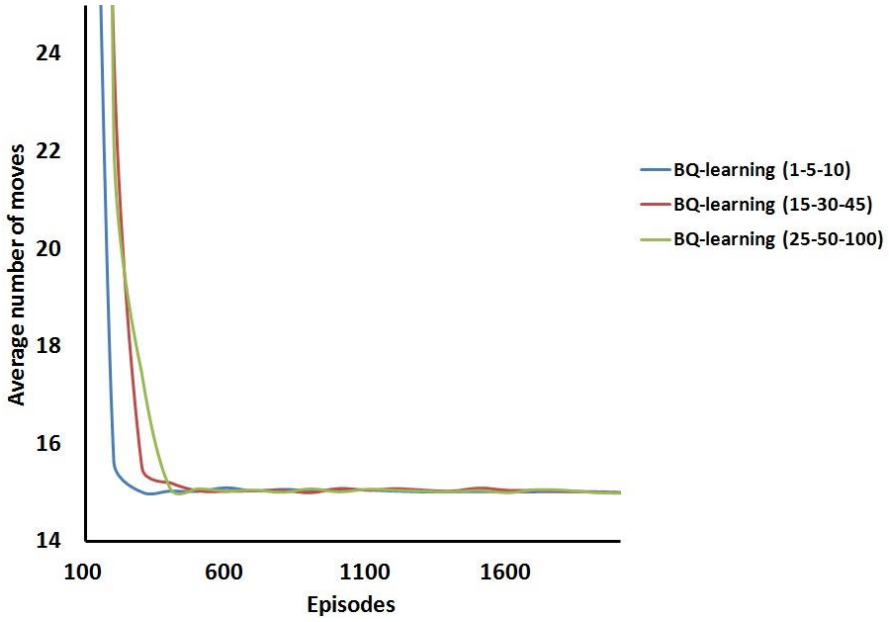

Figure 10. Experiment 5: Performance of different instances of BQ-learning in a shortest path problem of a grid size of 20×20 . Each curve is the average of 100 runs.

6. CONCLUSION AND FUTURE WORK

Cooperative Q-learning approach is an efficient learning approach that accelerates the learning process of individual learners in homogeneous multi-agent systems. This paper presented the BQ-learning algorithm which is a new cooperative Q-learning that is inspired from the bat algorithm. The learning process of BQ-learning comprises two stages. First, the individual learning stage, where each agent learns or improves its own policy by implementing the standard Q-learning algorithm. Second, the learning by interaction stage, where the learners share their Q-values among each other using a Q-value sharing strategy based on the bat

algorithm.The BQ-learning algorithm has many advantages. First, compared to current cooperative Q-learning algorithms, the BQ-learning algorithm can be implemented to single-task and multi-task problems, because optimizing the tasks of a learning problem using the bat algorithm improves the overall solution for the problem. Second, the bat sharing strategy in BQ-learning increases the possibility of finding the optimal Q-values, because it attempts to balance between the exploration and exploitation of actions using tuning techniques that control its parameters (frequencies, pulse emission rates and loudness of the potential solutions).

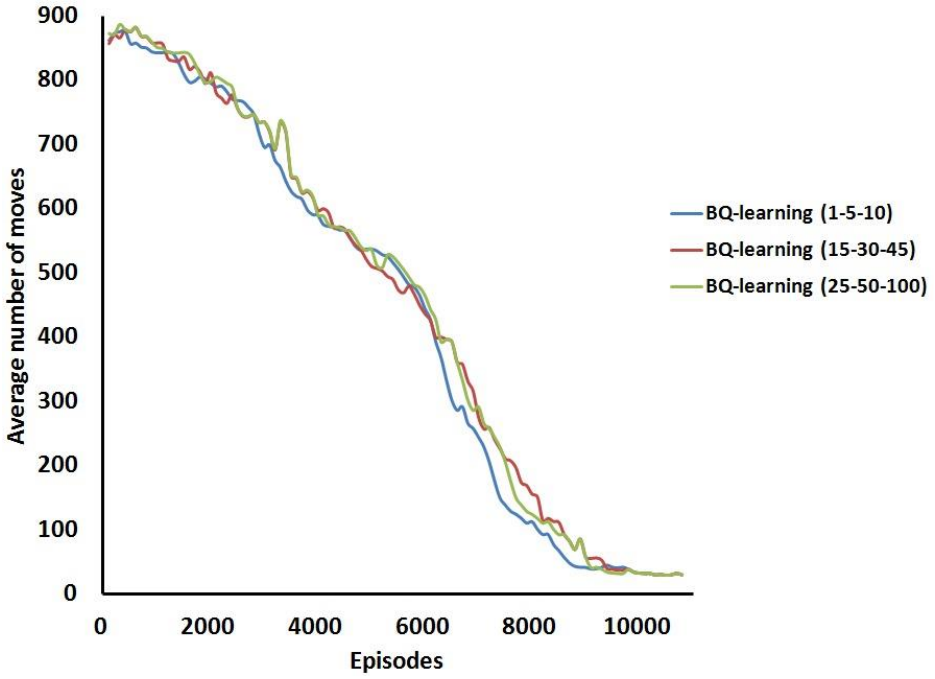

Figure 11. Experiment 6: Performance of different instances of BQ-learning in a taxi problem of a grid size of 5×5. Each curve is the average of 100 runs.

Finally, the results of the pilot experiments suggest that BQ-learning performs faster than single-agent Q-learning and other famous cooperative Q-learning algorithms, whether the agents have similar or different levels of experience and regardless of the type of the learning problems (single-task vs. multiple-task problems).

Future work includes implementing the BQ-learning algorithm to continuous space learning problems and developing a new cooperative Q-learning algorithm based on a combination of the firefly and monkey algorithms.

REFERENCES

[1] C. Watkins, Learning from Delayed Rewards, PhD thesis, Cambridge University, Cambridge, England, 1989.

[2] C. Watkins and P. Dayan, "Technical Note: Q-learning," Machine Learning, vol. 8, no. 3, pp. 279-292, 1992.

[3] B. H. Abed-alguni, S. K. Chalup, F. A. Henskens and D. J. Paul, "A Multi-agent Cooperative Reinforcement Learning Model Using a Hierarchy of Consultants, Tutors and Workers," Vietnam Journal of Computer Science, vol. 2, no. 4, pp. 213-226, 2015.

[4] P. Kormushev, S. Calinon and D. G. Caldwell, "Reinforcement Learning in Robotics: Applications and Real-world Challenges," Robotics, vol. 2, no. 3, pp. 122-148, 2013.

[5] J. Kober, J. A. Bagnell and J. Peters, "Reinforcement Learning in Robotics: A Survey," The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238-1274, 2013.

[6] H. Iima, Y. Kuroe and S. Matsuda, "Swarm Reinforcement Learning Method Based on Ant Colony Optimization," Proc. of 2010 IEEE International Conference on Systems, Man and Cybernetics (SMC), (O. Kaynak and G. Dimirovski, eds.), pp. 1726-1733, 2010.

[7] B. Cunningham and Y. Cao, "Non-reciprocating Sharing Methods in Cooperative Q-learning Environments," Proc. of the 2012 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, vol. 2, pp. 212-219, IEEE Computer Society, 2012.

[8] M. Tan, "Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents," Proc. of 10th International Conference on Machine Learning, vol. 337, Amherst, MA, 1993.

[9] B. H. Abed-alguni, D. J. Paul, S. K. Chalup and F. A. Henskens, "A Comparison Study of Cooperative Q-learning Algorithms for Independent Learners," International Journal of Artificial Intelligence, vol. 14, no. 1, pp. 71-93, 2016.

[10] H. Iima, Y. Kuroe and K. Emoto, "Swarm Reinforcement Learning Methods for Problems with Continuous State-action Space," Proc. of 2011 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 2173-2180, 2011.

[11] H. Iima and Y. Kuroe, "Reinforcement Learning through Interaction among Multiple Agents," Institute of Control, Automation and Systems Engineers (ICASE) and the Society of Instrument and Control Engineers (SICE) International Joint Conference, pp. 2457-2462, October 2006.

[12] H. Iima and Y. Kuroe, "Swarm Reinforcement Learning Algorithms-Exchange of Information among Multiple Agents," The Society of Instrument and Control Engineers (SICE), 2007 Annual Conference, pp. 2779-2784, September 2007.

[13] H. Iima and Y. Kuroe, "Swarm Reinforcement Learning Algorithms Based on SARSA Method," The Society of Instrument and Control Engineers (SICE) Annual Conference 2008, pp. 2045-2049, August 2008.

[14] E. Di Mario, Z. Talebpour and A. Martinoli, "A Comparison of PSO and Reinforcement Learning for Multi-robot Obstacle Avoidance," 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 149-156, June 2013.

[15] B. Dğan and T. Ölmez, "A Novel State Space Representation for the Solution of 2D-HP Protein Folding Problem Using Reinforcement Learning Methods," Applied Soft Computing, vol. 26, pp. 213-223, 2015.

[16] E. Pakizeh, M. Palhang and M. Pedram, "Multi-criteria Expertness Based Cooperative Q-learning," Applied Intelligence, vol. 39, no. 1, pp. 28-40, 2013.

[17] K.-S. Hwang, W.-C. Jiang and Y.-J. Chen, "Model Learning and Knowledge Sharing for a Multi-agent System with Dyna Q-learning," IEEE Transactions on Cybernetics, vol. 45, pp. 964-976, May 2015.

[18] M. N. Ahmadabadi and M. Asadpour, "Expertness Based Cooperative Q-learning," IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, vol. 32, no. 1, pp. 66-76, 2002.

[19] M. Ahmadabadi, A. Imanipour, B. Araabi, M. Asadpour and R. Siegwart, "Knowledge-based Extraction of Area of Expertise for Cooperation in Learning," International Conference on Intelligent Robots and Systems, 2006 IEEE/RSJ, pp. 3700-3705, 2006.

[20] X.-S. Yang, "A New Metaheuristic Bat-inspired Algorithm," Conference on Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), pp. 65-74, Springer, 2010.

[21] X.-S. Yang and A. Hossein Gandomi, "Bat Algorithm: A Novel Approach for Global Engineering Optimization," Engineering Computations, vol. 29, no. 5, pp. 464-483, 2012.

[22] X.-S. Yang, "Bat Algorithm for Multi-objective Optimization," International Journal of Bio-Inspired Computation, vol. 3, no. 5, pp. 267-274, 2011.

[23] J. Kennedy, "Particle Swarm Optimization," Encyclopaedia of Machine Learning, pp. 760-766, Springer, 2011.

[24] B. Hengst, "Model Approximation for HEXQ Hierarchical Reinforcement Learning," Machine Learning: ECML 2004 (J.-F. Boulicaut, F. Esposito, F. Giannotti and D. Pedreschi, eds.), vol. 3201 of Lecture Notes in Computer Science, pp. 144-155, Springer, Berlin, Heidelberg, 2004.

[25] T. G. Dietterich, "Hierarchical Reinforcement Learning With the MAXQ Value Function Decomposition," Journal of Artificial Intelligence Research, vol. 13, no. 1, pp. 227-303, 2000.

[26] D. Andre and S. J. Russell, "State Abstraction for Programmable Reinforcement Learning Agents," AAAI/IAAI, pp. 119-125, 2002.

[27] S. Ray and T. Oates, "Locking in Returns: Speeding Up Q-learning by Scaling," Proc. European Workshop on Reinforcement Learning (EWRL), pp. 32-36, 2011.

[28] R.-S. Sutton and A.-G. Barto, Reinforcement Learning: An Introduction, Cambridge, USA: MIT Press, 1998.

ملخص البحث:

يسمح منحنى تعلُّم كيو التعاوني لمتعلّمين متعدّدين بالتعلُّم بشكلٍ مستقلّ ومن ثمّ بتبادل قيم كيو الخاصة بهم فيما بينهم مستخدمين استراتيجية لتبادُل قيم كيو. وهناك مشكلة رئيسية ترتبط بهذا المنحنى، تتمثل في أن حُلول المتعلّمين قد لا تلتقي إلى المثالية؛ لأن قيم كيو المثالية قد لا يتمّ إيجادها. وهناك مشكلة أخرى هي أنّ بعض الخوارزميات التعاونية تكون ذات أداء جيّد فيما يتعلّق بالمشكلات ذات المهمّة الواحدة بينما يكون أداؤها ضعيفاً في المشكلات متعدّدة المهامّ.

تقترح هذه الورقة خوارزمية جديدة لتعلُّم كيو، وهي خوارزمية تعاونية تسمى خوارزمية الخفّاش، وتقوم بتنفيذ استراتيجية لتبادُل قيم كيو منبثقة عنها. والجدير بالذكر أنّ خوارزمية الخفّاش خوارزمية فعّالة تزيد من إمكانية إيجاد قيم كيو المثالية عن طريق إحداث التوازن بين الاستكشاف والاستغلال للأفعال عبر ضبط متغيرات الخوارزمية.

وقد تم اختبار خوارزمية الخفّاش لتعلُّم كيو باستخدام مسألتين هما: مسألة أقصر مسار، وهي مسألة أحادية المهمة، ومسألة التاكسي، وهي مسألة متعدّدة المهامّ. وتقترح النتائج التجريبية أنّ أداء خوارزمية الخفّاش لتعلُّم كيو أفضل عند مقارنته بأداء تعلُّم كيو ذي العامل الواحد وأداء بعض الخوارزميات التعاونية لتعلُّم كيو المعروفة جيداً.