Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Opposition-Based Q(λ) with Non-Markovian Update

Maryam Shokri[1] (Member, IEEE), Hamid R. Tizhoosh[1] (Member, IEEE), Mohamed S. Kamel[2] (Fellow, IEEE)
Pattern Analysis and Machine Intelligence Laboratory
[1]Department of Systems Design Engineering
[2]Department of Electrical and Computer Engineering
University of Waterloo, 200 University Avenue West, ON, N2L 3G1,a Canada
mshokri@engmail.uwaterloo.ca, tizhoosh@uwaterloo.ca, mkamel@uwaterloo.ca

Abstract— **The $OQ(\lambda)$ algorithm benefits from an extension of eligibility traces introduced as opposition trace. This new technique is a combination of the idea of opposition and eligibility traces to deal with large state space problems in reinforcement learning applications. In our previous works the comparison of the results of $OQ(\lambda)$ and conventional Watkins' $Q(\lambda)$ reflected a remarkable increase in performance for the $OQ(\lambda)$ algorithm. However the Markovian update of opposition traces is an issue which is investigated in this paper. It has been assumed that the opposite state can be presented to the agent. This may limit the usability of the technique to deterministic environments. In order to relax this assumption the Non-Markovian Opposition-Based $Q(\lambda)$ ($NOQ(\lambda)$) is introduced in this work. The new method is a hybrid of Markovian update for eligibility traces and non-Markovian-based update for opposition traces. The experimental results show improvements of learning speed for the proposed technique compared to $Q(\lambda)$ and $OQ(\lambda)$. The new technique performs faster than $OQ(\lambda)$ algorithm with the same success rate and can be employed for broader range of applications since it does not require determining state transition.**

I. Introduction

In $OQ(\lambda)$ algorithm [11] the idea of opposition has been implemented in the framework of Watkins' $Q(\lambda)$. Also, the problem of *reward/punishment confusion* which sometimes occurs in opposition-based Q-learning has been addressed [11]. In this technique the agent takes one action in a given state but updates the Q-values for all actions and opposite actions using the concept of eligibility and opposition traces. $OQ(\lambda)$ uses the advantage of multiple updating by opposition traces which is mostly applied for learning with delayed reward.

However the environment model is not always readily accessible for many decision problems. For instance, in many navigation problems, e.g. Mars Pathfinder, the model is not available and the robot should navigate in a completely unknown environment. Therefore, improvement of model-free techniques such as $Q(\lambda)$ plays an important role to solve such problems, especially concerning fast convergence rates. The next major goal of this research is to extend the usability of $OQ(\lambda)$ to a broader range of non-deterministic applications. This research is specifically focuses on investigating the possible Non-Markovian updating of opposition traces where the next state for opposite action may not be available. This extends the usability of $OQ(\lambda)$ to a broader range of applications where the model of environment is not provided for the agent.

The paper consists of seven sections. In Section II we describe opposition-based learning (OBL). A general overview of Q-Learning, $Q(\lambda)$, and Opposition-Based Q-learning OQ are presented in Section III. The previous extension of $Q(\lambda)$ with opposition concept, $OQ(\lambda)$, is presented in Section IV as well as the proposed technique with non-Markovian opposition update. In the Section V primary results are presented and compared with the results of Watkins' $Q(\lambda)$ and $OQ(\lambda)$. Section VI contains some conclusions, and directions for future work.

II. Opposition-Based Learning (OBL)

Opposition-based learning (OBL) has been recently introduced [11], [16], [17], [18]. The concept of OBL is new and not fully investigated yet.

When looking for a solution x to a given problem, we usually make estimate $\hat{x}$, which is not an exact solution but based on experience or on a totally random guess. Complex problems usually involve Guesses, e.g. random initialization of weights in a neural net. In some cases, the estimate $\hat{x}$ is sufficiently accurate while in others, we seek increased accuracy in results.

Reinforcement learning often begins randomly from scratch and moves, via exploration and exploitation, toward an existing solution (optimal policy). The action policy of reinforcement agents (which action to take at any given time and for any given state) is initially based on randomness. The random action, if leading to optimal state, may result in fast convergence. If the random guess however is far from the existing solution, e.g. worst-case situation, it is located in the *opposite direction*, search or optimization requires considerably more time, or may not be attained. Logically, we should look in all directions simultaneously, or, more concretely, in the opposite direction.

If we are searching for solution x and we agree that searching in the opposite direction could be advantageous, the first step becomes calculating the *opposite number* $\breve{x}$ [16].

Definition: Let $x \in R$ be a real number defined on a certain interval: $x \in [a, b]$. Opposite number $\breve{x}$ is defined as follows:

$$\breve{x} = a + b - x. \tag{1}$$

For $a = 0$ and $b = 1$, we receive

$$\breve{x} = 1 - x. \tag{2}$$

The opposite number in a multidimensional case is defined analogously.

Definition: Let $S(x_1, x_2, ..., x_n)$ be a point in $n-$dimensional coordinates with $x_1, x_2, ..., x_n \in R$ and $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, ..., n\}$. Opposite point $\breve{S}$ is completely defined by its coordinates $\breve{x_1}, \breve{x_2}, ..., \breve{x_n}$, where

$$\breve{x_i} = a_i + b_i - x_i. \tag{3}$$

The opposition-based scheme for learning now becomes concrete [16]:

> **Opposition-Based Learning:** Let $f(x)$ be the function in focus and $\hbar$ a proper evaluation function. If $x \in [a, b]$ is an initial (random) guess and $\breve{x}$ is its opposite, then in each iteration, we calculate $f(x)$ and $f(\breve{x})$. Learning continues with x if $\hbar(f(x)) \geq \hbar(f(\breve{x}))$, otherwise with $\breve{x}$. As a measure of optimality, evaluation function $\hbar(.)$ compares the suitability of results, e.g., fitness, reward and punishment, error, cost, or utility.

III. Q-Learning

Q-learning is off-policy TD (temporal differencing) control and is one of the most popular methods in reinforcement learning. In an off-policy technique the learned action-value function, $Q(s, a)$, directly approximates the optimal action-value function Q^*, independent of the policy being followed. In other words, the agent uses greedy policy as well as exploratory policy for action selection [15]. The agent learns to act optimally in Markovian domains by experiencing sequences of actions. The agent takes an action in a particular state and uses immediate reward and punishment to estimate the state value. It evaluates the consequences of taking different actions. By trying all actions in all states multiple times, the agent learns which action is best overall for each visited state [21]. The agent must determine an optimal policy and maximize the total (discounted) expected reward. Equation 4 is employed for updating the action-value function, Q, where s is the current state, a is action, α is the learning rate, γ is a discount factor, r is the reward or punishment, a' is the next action, and s' is the next state:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]. \tag{4}$$

The task of Q-learning is to determine an optimal policy π^*. The values of the Q matrix are the expected discounted reward for executing action a in state s based on policy π [21]. The (theoretical) condition for convergence of the Q-algorithm is that the sequence of episodes which forms the basis of learning must visit all states infinitely. It must be mentioned that based on Watkins and Dayan's Theorem in [21] the rewards and learning rate are bounded ($|r_n| \leq R_{max}$, $0 \leq \alpha_n < 1$).

A. $Q(\lambda)$ Technique

$Q(\lambda)$ has been considered as a bridge between Monte Carlo and Q-learning based on eligibility traces. The eligibility trace is one of the mechanisms in reinforcement learning [15]. The idea is that only eligible states or actions will be assigned a credit or blamed for an error. Sutton et al. [15] describe an eligibility trace as "a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action". Eligibility traces require more complex implementation but yield faster learning especially for applications with delayed rewards.

The Watkins' $Q(\lambda)$ looks ahead until the next exploratory action. If the agent takes an exploratory action, then the eligibility traces will become zero. During the exploration process in early learning the traces will be cut off and as a result learning reward will not be delayed until the end of learning episode [15].

In this research the Watkins' $Q(\lambda)$ has been implemented (see Table I). We explain the opposition-based Q-learning (OQ) to establish a basic understanding. Then in the next section we demonstrate the extension of $Q(\lambda)$ algorithm based on the opposition concept in order to introduce its opposition-based extension $OQ(\lambda)$.

B. OQ: Opposition-Based Q-learning

How can the idea of opposition and RL be combined? For this purpose, at each time t, if the agent receives a reward for taking action a in a given state s, then the agent may also receive punishment for opposite action $\breve{a}$ in the same state s without taking the opposite action. It means that the value function, Q, (e.g. Q-matrix in tabular Q-learning) can be updated for two values $Q(s, a)$ and $Q(s, \breve{a})$ instead of only one value $Q(s, a)$. Therefore, agent can simultaneously explore actions and opposite actions. Hence, updating the Q-values for two actions in a given state for each time step can lead to faster convergence since the Q-matrix can be filled in a shorter time [17], [16], [18].

Figure 1 demonstrates the difference between Q-matrix updating using reinforcement learning (left image) and Q-matrix updating using opposition-based reinforcement learning (right image).

IV. Proposed technique

In this work, we propose an improvement of $OQ(\lambda)$ [11] by introducing a new way of updating the opposition trace. This new update is independent of determining the next state of the environment for opposite action. In the following $OQ(\lambda)$ technique is reviewed. Then $NOQ(\lambda)$ algorithm is introduced and the details are discussed.

A. $OQ(\lambda)$: Opposition-based $Q(\lambda)$

The first extension of the concept of opposition to $Q(\lambda)$ was introduced by using *opposition traces* [11]. The eligibility traces for opposite actions are called opposition traces. Assume that $e(s, a)$ is the eligibility trace for action a in state s, then the opposition trace is $\breve{e} = e(s, \breve{a})$. For updating

TABLE I

TABULAR WATKINS' $Q(\lambda)$ ALGORITHM [15]

```
For all $s$ and $a$ initialize $Q(s,a)$ with arbitrary numbers and initialize $e(s,a) = 0$
For each episode repeat
    Initialize $s$ and $a$
    For each step of episode repeat
        Take action $a$, observe $r$ and next state $s'$
        Choose next action $a'$ from $s'$ using policy
        $a^* \longleftarrow argmax_b\ Q(s',b)$ (if $a'$ ties for max, then $a^* \leftarrow a'$)
        $\delta \longleftarrow r + \gamma\ Q(s',a^*) - Q(s,a)$
        $e(s,a) \longleftarrow e(s,a) + 1$
        For all $s$,$a$
            $Q(s,a) \longleftarrow Q(s,a) + \alpha\delta e(s,a)$
            If $a' = a^*$, then $e(s,a) \longleftarrow \gamma\lambda e(s,a)$
            else $e(s,a) \longleftarrow 0$
        $s \longleftarrow s'$; $a \longleftarrow a'$
    until $s$ is terminal
```

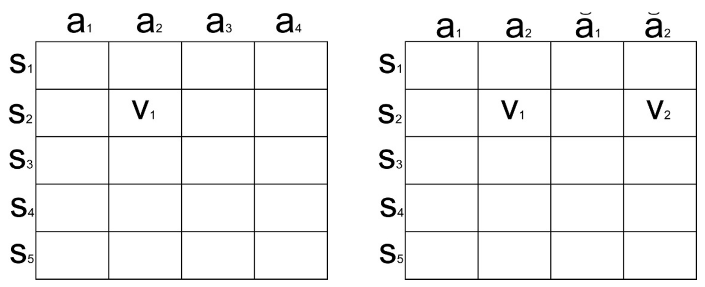
a_1 a_2 a_3 a_4
s_1 s_2 s_3 s_4 s_5
v_1
a_1 a_2 $\breve{a}_1$ $\breve{a}_2$
s_1 s_2 s_3 s_4 s_5
v_1
v_2

Fig. 1. Q-matrix updating; Left: Q-matrix updating using reinforcement learning, right: Q-matrix updating using opposition-based reinforcement learning where additional update (v_2) can be performed for the opposite action $\breve{a}_2$.

in a given state s, agent takes action a and receives reward r. Then by using reward r the Q-matrix will be updated for all states s and actions a:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha\delta_1 e(s,a), \qquad (5)$$

where δ_1 is presented in Table II, and $e(s,a)$ is eligibility traces for states s and actions a. By assuming that the agent will receive punishment p by taking opposite action $\breve{a}$ in state s, the opposition trace can be updates. The formula for updating the Q-values is presented as follows:

$$Q(s,\breve{a}) \longleftarrow Q(s,\breve{a}) + \alpha\delta_2 e(s,\breve{a}), \qquad (6)$$

where δ_2 is presented in Table II, and $e(s,\breve{a})$ is the opposition traces for states s and opposite actions $\breve{a}$. The algorithm of the $OQ(\lambda)$ technique is presented in Table II where the opposite reward $\breve{r}(= p)$ is punishment.

The opposition trace acts as a facilitator for updating the Q-values for opposite actions to benefit from the idea of concurrent updating for actions and opposite actions.

The advantage of using concurrent updating in $OQ(\lambda)$ is that instead of punishing/rewarding the action and opposite action we punish/reward the eligible trace and opposite trace. It is assumed that when the agent receives reward (r)/punishment ($\breve{r}$) for taking an action, it will receive punishment ($\breve{r}$)/reward (r) for taking the opposite action.

The other advantage of opposition trace is solving the problem of *Reward/Punishment Confusion*. This problem usually occurs in the situations when action and opposite action in a given state may yield the same result instead of opposite results. The action and opposite action may both lead to reward, or both lead to punishment. The example in Figure 2 [11] illustrates this situation in the grid world problem. The goal is presented by a star and the present state is s. For both action a_1, and its opposite $\breve{a}_1$ the result

TABLE II

$OQ(\lambda)$ ALGORITHM. IF THE AGENT RECEIVES PUNISHMENT p FOR TAKING AN ACTION THEN THE OPPOSITE ACTION RECEIVES REWARD r. LINES INDICATED BY ASTERISKS (*) EXTEND $Q(\lambda)$ TO $OQ(\lambda)$ [11]

```
For all $s$ and $a$ initialize $Q(s,a)$ with arbitrary numbers and initialize $e(s,a) = 0$
For each episode repeat
    Initialize $s$ and $a$
    For each step of episode repeat
        Take action $a$, observe $r$ and next state $s'$
      * Determine opposite action $\breve{a}$ and next state $s''$
      * Calculate opposite reward (punishment) $\breve{r} = p$
        Choose next action $a'$ from $s'$ using policy
      * Determine next opposite action $\breve{a}'$ from $s''$
        $a^* \longleftarrow argmax_b\ Q(s', b)$ (if $a'$ ties for max, then $a^* \leftarrow a'$)
      * $a^{**} \longleftarrow argmax_b\ Q(s'', b)$ (if $\breve{a}'$ ties for max, then $a^{**} \leftarrow \breve{a}'$)
        $\delta_1 \longleftarrow r + \gamma\ Q(s', a^*) - Q(s, a)$
      * $\delta_2 \longleftarrow \breve{r} + \gamma\ Q(s'', a^{**}) - Q(s, \breve{a})$
        $e(s, a) \longleftarrow e(s, a) + 1$
      * $e(s, \breve{a}) \longleftarrow e(s, \breve{a}) + 1$
        For all $s$, $a$
          $Q(s, a) \longleftarrow Q(s, a) + \alpha\delta_1 e(s, a)$
          If $a' = a^*$, then $e(s, a) \longleftarrow \gamma\lambda e(s, a)$
          else $e(s, a) \longleftarrow 0$
        $s \longleftarrow s'$; $a \longleftarrow a'$
      * For all $s$, $\breve{a}$:
        * $Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + \alpha\delta_2 e(s, \breve{a})$
        * If $\breve{a}' = a^{**}$, then $e(s, \breve{a}) \longleftarrow \gamma\lambda e(s, \breve{a})$
        * else $e(s, \breve{a}) \longleftarrow 0$
    until $s$ is terminal
```

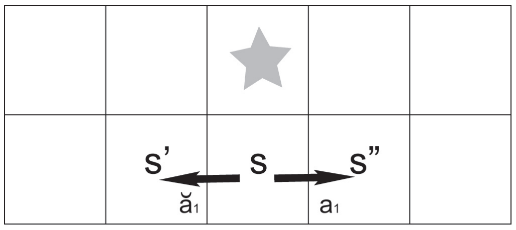

Fig. 2. Example to show that action and opposite action in state s can yield the same result, here punishment p. The target is presented by a star and the next states are s' or s'' respectively

is punishment because they both increase the distance of the agent from the goal. Hence, both of them should be punished. Rewarding one of them will falsify the value function and affect the convergence.

B. $NOQ(\lambda)$: Opposition-based $Q(\lambda)$ with Non-Markovian Update

One issue regarding the $OQ(\lambda)$ algorithm is the problem of Markovian-based updating of the opposite trace which will be addressed here. As it is presented in the algorithm of $OQ(\lambda)$ (Table II, line 6), the agent should determine the next state s'' after defining the opposite action $\breve{a}$. In this algorithm the agent does not actually take $\breve{a}$, but rather determines the opposite one. For instance if the action is going to the left in the Gridworld [11] then the opposite action is determined (not taken) as going to the right. In a deterministic environment the agent can figure out the next state by using the model of the environment. In the case of Gridworld problem the agent assumes that the next state s'' for opposite action $\breve{a}$ is the opposite of next state $\breve{s}'$ with respect to initial states. In this case the Q values can be updated for the opposition trace as follows:

$$Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + \alpha\delta_2 e(s, \breve{a}), \tag{7}$$

where

$$\delta_2 \longleftarrow \breve{r} + \gamma Q(s'', a^{**}) - Q(s, \breve{a}). \tag{8}$$

Equations 7 and 8 present the Markovian-based updating by considering the next state of s''. In $NOQ(\lambda)$ we address this problem by introducing the non-Markovian updating for opposition traces. In the following paragraphs the details and the algorithm of the proposed technique are introduced.

As it is mentioned earlier the $OQ(\lambda)$ method updates opposite traces without taking opposite actions. For this reason the opposition update (the Markovian update) depends on the agent that should know the next state of the opposite action. This limits the applicability of the $OQ(\lambda)$ to the deterministic environments. We relax the constraint of Markovian updating by introducing a new update for opposition traces in $NOQ(\lambda)$. Equation 9 presents the new update formula for opposition traces where W is opposition weight, $\breve{r}$ is the opposite reward, and $e(s, \breve{a})$ is the opposite trace.

$$Q(s, \breve{a}) \longleftarrow Q(s, \breve{a}) + W * \breve{r} * e(s, \breve{a}). \tag{9}$$

W is a parameter introduced to impose a weight between 0 and 1 to the opposition update. If in some application the definition of opposite action is not straightforward then we assume that at the beginning of the learning the weight of the update is low and increases gradually as the agent explores the actions and the opposite actions. In the case of the Gridworld problem which will be introduced in the next section the definition of opposite actions are known and the weight W is set to 1.

As it is presented in the Equation 9 the $Q(s, \breve{a})$ in the proposed method does not depend on the next state in contrast to the $OQ(\lambda)$ technique which the update depends on the s'' (see Equations 7 and 8). The algorithm of the proposed technique is presented in the Table III.

Figure 3 is $NOQ(\lambda)$ backup diagram which is an extension of the backup diagram for Watkins' $Q(\lambda)$ presented by Sutton et al. [15]. There are two backups, one based on eligibility traces and the other one is for opposition traces.

Figure 4 is also the extension of the Gridworld example of eligibility trace presented by Sutton et al. [15]. In this extension the Gridworld is presented with two traces. The black arrows represent increases of the action values. On the other hand, the gray arrows with different sizes represent the negative increases (decreases) in the action values for the opposition trace. The trace ends at a location of high reward value marked by a star.

V. Experimental Results

The Gridworld problem with three sizes (20×20, 50×50, and 100×100) is chosen as a test-case problem. The grid represents the learning environment and each cell of the grid represents a state of the environment. A sample Gridworld is presented in Figure 5. The agent can move in 8 possible directions indicated by arrows in the figure. The goal of the agent is to reach the defined target in the grid which is marked by a star.

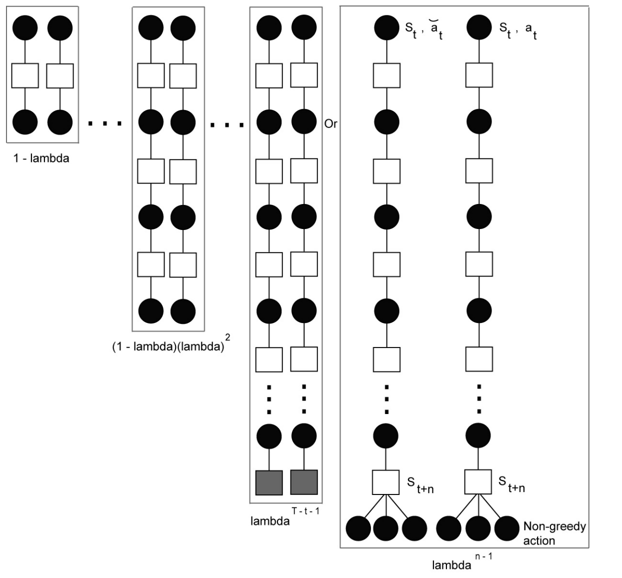

Fig. 3. The backup diagram for $NOQ(\lambda)$ which is an extension of backup diagram for Watkins's $Q(\lambda)$ similar to the diagram presented by Sutton et al. [15]

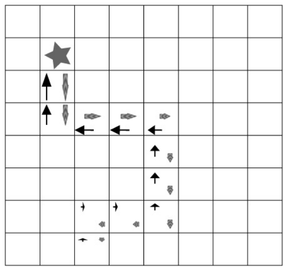

Fig. 4. The extension of the Gridworld with eligibility trace similar to the example presented by Sutton et al. [15]. In this extension the Grid world is presented with two traces, eligibility trace and opposition trace

Four actions with their corresponding four opposite actions (if $a_l = Up$ then $\breve{a}_l = Down$) are defined [11]. By taking an action agent has the ability to move to one of the neighboring states. The actions/opposite actions are left/right, up/down, up-right/down-left, and up-left/down-right. The initial state is selected randomly for all experiments. If the size of a grid is (X_{max}, Y_{max}), then the coordinates of the target is fixed at $(\frac{X_{max}}{2}, \frac{Y_{max}}{3})$. The value of immediate reward is 10, and punishment is -10. After agent takes an action, if the distance of the agent from the goal is decreased, then agent will receive reward. If the distance is increased or not changed, the agent receives punishment. The Boltzmann policy [5] is applied for all implementations. The $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms are implemented. The initial parameters for the algorithms are presented in Table IV.

The following measurements are considered for comparing the results of $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms:

- overall average iterations $\overline{\overline{I}}$: average of iterations over 100 runs

TABLE III

$NOQ(\lambda)$ ALGORITHM. IF THE AGENT RECEIVES PUNISHMENT p FOR TAKING AN ACTION THEN THE OPPOSITE ACTION RECEIVES REWARD r. LINES INDICATED BY ASTERISKS (*) EXTEND $Q(\lambda)$ TO $OQ(\lambda)$ AND LINES INDICATED BY ASTERISKS (**) EXTEND $OQ(\lambda)$ TO $NOQ(\lambda)$

```
For all $s$ and $a$ initialize $Q(s,a)$ with arbitrary numbers and initialize $e(s,a) = 0$
For each episode repeat
    Initialize $s$ and $a$
    For each step of episode repeat
        Take action $a$, observe $r$ and next state $s'$
      ** Determine opposite action $\breve{a}$
        Choose next action $a'$ from $s'$ using policy
      ** Determine next opposite action $\breve{a}'$
        $a^* \longleftarrow argmax_b\ Q(s',b)$ (if $a'$ ties for max, then $a^* \leftarrow a'$)
        $\delta_1 \longleftarrow r + \gamma\ Q(s',a^*) - Q(s,a)$
        $e(s,a) \longleftarrow e(s,a) + 1$
      * $e(s,\breve{a}) \longleftarrow e(s,\breve{a}) + 1$
        For all $s$, $a$
            $Q(s,a) \longleftarrow Q(s,a) + \alpha\delta_1 e(s,a)$
            If $a' = a^*$
                $e(s,a) \longleftarrow \gamma\lambda e(s,a)$
                $e(s,\breve{a}) \longleftarrow \gamma\lambda e(s,\breve{a})$
            else
                $e(s,a) \longleftarrow 0$
              ** $e(s,\breve{a}) \longleftarrow 0$
        $s \longleftarrow s'$; $a \longleftarrow a'$
      ** For all $s$, $\breve{a}$:
        ** $Q(s,\breve{a}) \longleftarrow Q(s,\breve{a}) + W * \breve{r} * e(s,\breve{a})$
    until $s$ is terminal
```

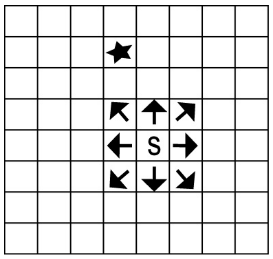

Fig. 5. Sample Gridworld. There are eight possible actions presented by arrows. S is a state and the star is the target

- average time $\overline{T}$: average of running time (seconds) over 100 runs
- number of failures χ: number of failures in 100 runs

The agent performs the next episode if it reaches the target represented by star. The learning stops when the accumulated reward of the last 15 iterations has a standard deviation below 0.5. The results are presented in Table V.

The results presented in the Table V are also plotted in Figures 6 and 7 for visual comparisons.

Figure 6 presents changes of overall average iterations for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms for the three grids, 20×20, 50×50 and 100×100 over 100 runs. We observe that the total number of iterations for convergence of $Q(\lambda)$ is far higher than $OQ(\lambda)$ and $NOQ(\lambda)$ algorithms. The $NOQ(\lambda)$ takes slightly less iterations than $OQ(\lambda)$.

TABLE IV

THE INITIAL PARAMETERS FOR ALL EXPERIMENTS

n_E	I_{max}	α	γ	λ
100	1000	0.3	0.2	0.5

Figure 7 presents average time for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ algorithms for the three grids, 20×20, 50×50 and 100×100 over 100 runs. Even though the number of iterations for $OQ(\lambda)$ and $NOQ(\lambda)$ are almost the same but the average computation time of $NOQ(\lambda)$ is much less than the average time of the $OQ(\lambda)$ for 100×100 grid. The reason is that $NOQ(\lambda)$ algorithm is more efficient than $OQ(\lambda)$ due to decrease in the computational overhead associated with updating the opposition traces.

In order to compare $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$ al-

TABLE V

THE RESULTS FOR THE FIVE MEASURES OF $\overline{\overline{I}}$, $\overline{E}$, $\overline{T}$, χ, AND ζ FOR ALGORITHMS ($Q(\lambda)$, $OQ(\lambda)$, AND $NOQ(\lambda)$). THE RESULTS ARE BASED ON 100 RUNS FOR EACH ALGORITHM

	$Q(\lambda)$			$OQ(\lambda)$			$NOQ(\lambda)$		
$X \times Y$	20×20	50×50	100×100	20×20	50×50	100×100	20×20	50×50	100×100
$\overline{\overline{I}}$	255 ± 151	528 ± 264	431 ± 238	20 ± 8	57 ± 26	103 ± 50	19 ± 7	48 ± 23	100 ± 46
$\overline{T}$	3 ± 2	17 ± 6.6	102 ± 25	0.3 ± 0.1	5 ± 1	80 ± 7	0.2 ± 0.1	5 ± 0.7	58 ± 6
χ	0	11	9	0	0	0	0	0	0
ζ	93.3%			100%			100%		

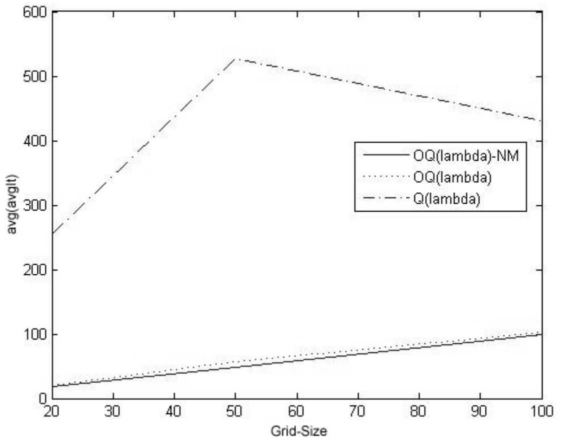

Fig. 6. The changes of average of average iterations $\overline{I}$ for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$

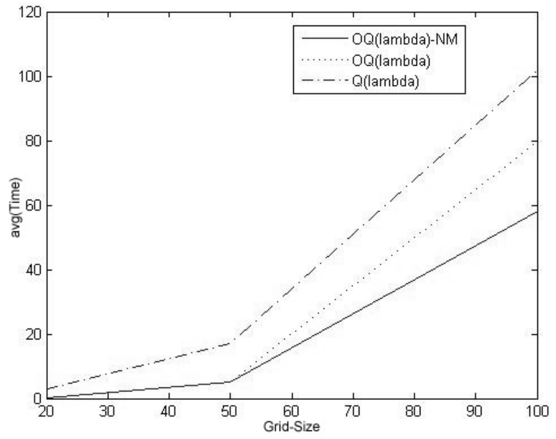

Fig. 7. The changes of average time T for $Q(\lambda)$, $OQ(\lambda)$, and $NOQ(\lambda)$

gorithms we also need to consider that $Q(\lambda)$ failed[1] to reach the goal or target 20 times. To reflect this failure in the performance measure, the success rate $\zeta_{overall}$ for the algorithms (presented in [11]) is calculated:

$$\zeta_{overall} = (1 - \frac{\sum_{i=1}^{k} \chi_i}{\sum_{k} H}) \times 100, \tag{10}$$

where k is the number of grids tested (in this case $k = 3$), χ is the number of failures, and H is the number of times the code is run for each grid. Considering the convergence conditions, for the $Q(\lambda)$ algorithm the overall success rate is $\zeta_{overall} = 93.3\%$ because the agent failed to reach the goal 20 times. For the proposed algorithm $NOQ(\lambda)$, and the oppsotion-based algorithm $OQ(\lambda)$, the overall success rate is $\zeta_{overall} = 100\%$; They always successfully find the target and reach the goal.

[1]If the fixed numbers of iterations and episodes are not enough for the algorithm to reach the target (in one run) then we consider this as one failure.

VI. CONCLUSIONS

In previous work [11], $OQ(\lambda)$ was introduced to accelerate $Q(\lambda)$ algorithm with discrete state and action space. This research is an extension of $OQ(\lambda)$ to a broader range of non-deterministic environments. The update of opposition trace in $OQ(\lambda)$ depends on next state of the opposite action (which cannot be taken). This limits the usability of this technique to the deterministic environments because the next state should be detected or known by the agent.

In this paper an extended version, $NOQ(\lambda)$, was presented to update the opposition traces independent of knowing the next state for the opposite trace. The primary results show that $NOQ(\lambda)$ can be employed in non-deterministic environments and performs even faster (see Figure 7) than $OQ(\lambda)$.

Generally, defining opposite actions is not always a straightforward task. The applicability of learning opposite actions through the exploration and defining a dynamic procedure for adapting the opposition weight W will be considered in our future work.

APPENDIX

Nomenclature for the Presented Results

- α: Learning step
- γ: Discount factor
- λ: Parameter for eligibility traces
- $X_{max} \times Y_{max}$: Grid size
- n_E: Maximum number of episodes
- I_{max}: Maximum number of iterations
- T: Running time of the program
- $\overline{\overline{I}}$: Average of iterations over 100 runs (overall average iterations)
- E: Episodes taken to reach the convergence criteria
- $\overline{E}$: Average of episodes over 100 times run
- $\overline{T}$: Average of running time (seconds) over 100 times run
- ζ: Success rate
- H: Number of times the code is run (the algorithm with all episodes and iterations) for a specific grid size which is 100 times for each of the grids, 20×20, 50×50 and 100×100
- χ: Number of failures
- G: Grid
- W: opposition weight

REFERENCES

[1] A. Ayesh, Emotionally Motivated Reinforcement Learning Based Controller, proceedings of IEEE SMC conference, The Hague, The Netherlands, page(s): 874- 878, vol.1, 2004

[2] S. Gadanho, Reinforcement Learning in Autonomous Robots: An Empirical Investigation of the Role of Emotions, PhD Thesis, University of Edinburgh, Edinburgh, 1999

[3] S. K. Goel, Subgoal Discovery for Hierarchical Reinforcement learning Using Learned Policies, Master of Science in Computer Science and Engineering, Department of Computer Science and Engineering, University of Texas at Arlington, TX, USA, 2003

[4] C. L. Isbell Jr., C. R. Shelton, M. Kearns, S. Singh, P. Stone, Cobot: A Social Reinforcement Learning Agent, Advances in Neural Information Processing Systems 14 (NIPS), 1393-1400, 2002

[5] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research 4, 237-285, 1996

[6] L. Paletta, E. Rome, Reinforcement Learning for Object Detection Strategies, Proceedings of 8th International Symposium on Intelligent Robotic Systems, SIRS 2000, University of Reading, UK, 2000

[7] J. Peng, B. Bhanu, Learning to Perceive Objects for Autonomous Navigation, Autonomous Robots 6, 187-201, 1999

[8] P. Preux, S. Delepoulle, J. C. Darcheville, A generic Architecture for Adaptive Agents Based on Reinforcement Learning, Information Sciences-Informatics and Computer Science: An International Journal, Elsevier Science Inc., vol. 161, Issue 1-2, 37-55, 2004

[9] C. Ribeiro, Reinforcement Learning Agent, Artificial Intelligence Review, 17, 223-250, 2002

[10] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Pearson Education Inc., New Jersey, 2003

[11] M. Shokri, H. R. Tizhoosh, M. Kamel, Opposition-Based Q(lambda) Algorithm, International Joint Conference on Neural Networks, IJCNN, 646-653, 2006

[12] M. Shokri, H. R. Tizhoosh, $Q(\lambda)$-Based Image Thresholding, 1st Canadian Conference on Computer and Robot Vision (CRV 2004), 504-508, 2004

[13] M. Shokri, H. R. Tizhoosh, Using Reinforcement Learning for Image Thresholding, Canadian Conference on Electrical and Computer Engineering, 1, 1231-1234, 2003

[14] S. Singh, D. Litman, M. Kearns, M. Walker, Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System, Journal of Artificial Intelligence (JAIR), Vol. 16, 105-133, 2002

[15] R. S. Sutton, A.G. Barto, Reinforcement learning: An Introduction, Cambridge, Mass., MIT Press, 1998

[16] H. R. Tizhoosh, Opposition-Based Learning: A New Scheme for Machine Intelligence, International Conference on Computational Intelligence for Modeling Control and Automation - CIMCA'2005, Vienna, Austria, vol. I, 695-701, 2005

[17] H. R. Tizhoosh, Reinforcement Learning Based on Actions and Opposite Actions, ICGST International Conference on Artificial Intelligence and Machine Learning (AIML-05), Cairo, Egypt, 2005

[18] H. R. Tizhoosh, Opposition-Based Reinforcement learning, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 10, No. 4, 578-585, 2006

[19] H. R. Tizhoosh, M. Shokri, M. Kamel, The Outline of a Reinforcement-Learning Agents for E-Learning Applications, Accepted for Samuel Pierre (ed.), E-Learning Networked Environments and Architectures: A Knowledge Processing Perspective, Springer Book Series, 2005

[20] C. J. C. H. Watkins, Learning from Delayed Rewards, PhD Thesis, Cambridge, Cambridge University, 1989

[21] C. J. H. Watkins, P. Dayan, Technical Note, Q-Learning, Machine Learning, 8, 279-292, 1992