A Hybrid Learning Strategy for Discovery of Policies of Action

Richardson Ribeiro, Fabrício Enembreck, Alessandro L. Koerich

Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Pontifícia Universidade Católica do Paraná
Rua Imaculada Conceição, 1155, CEP 80215-901
Curitiba, Paraná – Brasil
{richard,alekoe,fabricio}@ppgia.pucpr.br

Abstract. This paper presents a novel hybrid learning method and performance evaluation methodology for adaptive autonomous agents. Measuring the performance of a learning agent is not a trivial task and generally requires long simulations as well as knowledge about the domain. A generic evaluation methodology has been developed to precisely evaluate the performance of policy estimation techniques. This methodology has been integrated into a hybrid learning algorithm which aim is to decrease the learning time and the amount of errors of an adaptive agent. The hybrid learning method namely K-learning, integrates the Q-learning and K Nearest-Neighbors algorithm. Experiments show that the K-learning algorithm surpasses the Q-learning algorithm in terms of convergence speed to a good policy.

1 Introduction

Reinforcement Learning (RL) is a computational paradigm of learning where an algorithm attempts to maximize a measure of performance based on the reinforcements (reward or punishment) that it receives when interacting with an unknown environment [11]. In the literature one can find several approaches to RL and examples of different applications using intelligent agents [8] [10] [11] [14] [18] [19].

In an attempt to optimize the learning task of an agent, many authors have integrated different learning methods, creating the so-called hybrid learning methods, with the aim that one can compensate the weaknesses of each other, leading to more robust agents with higher tolerance to faults [4] [7] [15] [17]. However, the evaluation of the action policies produced by these hybrid methods is a complex task due to the lack of generic mechanisms that allow measuring the performance of a learning agent without relaying on the knowledge of the problem domain (or independently of the problem domain). Current evaluation mechanisms are not generic enough to evaluate action policies in different environments.

In this paper we propose a novel hybrid learning method, unifying RL and instance-based learning algorithms. The aim of such a hybrid method is to discover good action policies in a short timeframe. Furthermore, we propose a novel generic methodology to evaluate the quality of the policy discovered by RL-based adaptive autonomous agents. This methodology allows the estimation of agents' performance without relaying on

the knowledge of the domain and in short simulation periods. In the proposed methodology, the measure of performance of an agent is proportional to the number of correct decisions produced by its action policy in a given environment. A policy represents a space of states with initial and target states where the transition between states is determined by the actions. Therefore, a correct decision is reached when the agent finds a way to reduce the cost between the initial and target states. When this occurs for all candidate states, an optimal policy was discovered. The proposed evaluation methodology allows observing the behavior of the learning algorithm as a function of the number of iterations, configuration of environment and values of parameters related to the algorithm. The development of a hybrid learning mechanism was possible because of these above mentioned characteristics of the evaluation methodology.

This article is organized as follows: Section 2 presents some hybrid learning methods. Section 3 describes the evaluation mechanisms employed in different classes of problems. The novel performance evaluation methodology as well as an overview of the Q-learning, the K-NN, and the K-learning algorithms are presented in Section 4. Experimental results are presented in Section 5. Finally, in the last section some conclusions are drawn as well as perspectives for future research.

2 Hybrid Learning Methods

Different hybrid learning methods have been proposed in the literature with the aim of integrating different learning methods. Henderson et al. [7] proposes a hybrid model that combines RL with supervised learning. RL is used to optimize the average of rewards between the communication of systems that use huge data sets and that also have large state spaces. The supervised learning is used to constrain the learned policy of one part of state's space and modeling a policy with the data used in a given moment. Downing [4] proposes a hybrid technique that combines RL with trees based on genetic programming. This technique adds a new element to a set of functions of the genetic programming and produces a system, whose actions are strengthened by genetic programming so that the successive functioning of some trees shows the improvement of task adjustment performance. Figueiredo et al. [6] propose a hybrid neuro-fuzzy model based on RL. This model presents important characteristics as: automatically learning the model structure; self-adjusting to the parameter performance associated with the structure; and capacity to learn actions that must be taken when the agent is in a state of the environment. RL is used in this hybrid model to determine more properly actions to be executed for a given state. Rayan [15] proposes a system that incorporates techniques of symbolic planning with RL with the aim of producing a system capable of intensifying each method. The system uses a new behavior representation, which is defined in terms of desired consequences, but letting the implementation of policy to be learned by RL.

We use some concepts presented in such works to develop a hybrid method, namely *K-learning,* that integrates two well-know learning techniques used in adaptive autonomous agents: RL (Q-learning algorithm) and Instances-Based Learning (K-NN algorithm). This hybrid method emphasizes the strengths of each learning algorithm:

the K-NN allows that states with similar characteristics have similar rewards, anticipating rewards and decreasing the number of iterations to the Q-learning. On the other hand, the Q-learning guarantees that an optimal policy be found along the iterations.

3 Policy Evaluation Mechanisms

Measuring performance of a learning agent is not a trivial task. It can be measured according to its proposed task and mainly according the desired objective. For such an aim, some measures such as speed of convergence for the optimal or almost-optimal behavior, total reinforcements obtained by the agent and percentage of optimality have been used [8].

Ernst et al. [5] present a metric of quality of solution produced by an agent, as a mechanism to evaluate the performance of RL algorithms. A stationary policy is used to calculate the expected values which are further compared with regression algorithms on a set of examples. These examples have initial states chosen from a set of vectors generated by the Q-learning algorithm and the average expected value of the stationary policy is computed. Ramon [12] used decision trees to find an approximated function Q to guarantee the convergence to an optimal policy. He proposes a learning algorithm that generates a new estimate Q' from a preceding estimate Q. The author uses three types of measures to evaluate the performance of an algorithm: an amount $Q'(s,a)$ up to date that is wanted; an estimate $\tilde{q}(s,a)$ of Q' that is measured in exploration executed by algorithm; a measure $\tilde{q}(c)$ where $c \subseteq SXA$ is an average of function $\tilde{q}$ on an abstraction c, where S is the set of states and A the set of action. The algorithm stores in memory the values of the third metric of form to generate statistics. Lev et al. [9] use heuristic value function that evaluates the performance of an agent by computing the states foreseen for $\tilde{v}*$ and selects an operator who leads to the most promising state being $\tilde{v}*$ the optimal value of the function.

Generally, the evaluation of policies is not flexible enough to deal with different kind of problems. Moreover, they require too much knowledge on the domain to define heuristics related to the problem solution. Due to this fact, there is a lack of generic evaluation mechanisms that can be used in problems that involve RL.

4 A Novel Evaluation Methodology

The novel evaluation methodology analyzes the performance of RL algorithms assuming that there is a reward function capable of reproducing an optimal action policy. We consider that the performance of an agent is proportional to the number of correct decisions produced by its action policy in a given environment. A policy represents state space with initial and target states, where the transition between states is determined by actions. Therefore, a correct decision is attained when the agent finds the path between the initial and target state which has the lowest cost. Otherwise, an error is found. When this occurs for all candidate states, it can be stated that an *optimal*

policy was discovered. In this way, the evaluation of a policy is carried out through a problem solving algorithm capable of finding the optimal path between two states where the heuristic function corresponds to the values returned by the reward function. Moreover, the amount of states visited can also be taken into account.

We use the *A Star* algorithm (*A**) to find the best solution which is complete and optimal [14]. Since the *A** produces the best policy for a given heuristic, the quality of the policy discovered by different learning algorithms can be stated through the comparison of the policy found by them against the policy produced by the *A**. This methodology shows the proximity between the current policy of the agent and the optimal policy. A correct decision occurs when the agent is able to reach the target with an optimal cost (shortest path). We use the number of states and the additional cost associated with each state (value of reward function) to evaluate the total cost of a path.

To evaluate the performance of learning algorithms, we place the agent in an unknown environment and adjust the parameters for each possible initial state. Further, we set the learning parameters of the agent. Thus, it is possible to analyze the performance of algorithms through the learning curves.

```
Algorithm EvaluateAlgoritm_X(PQ,PX)//PQ is action policy of the Q-
learning algorithm and PX is action policy of the X algorithm;
1 For a given environment PQ=0, PX=0;
2 For each iteration of the Q-learning repeat:
      Efficiency_Q_Learning ← evaluatePolicy(PQ)
         If stop_condition() = false then
             Return to 2
         Else
             Go to step 3
3 For each state s learned by the Q-learning repeat:
             PX ← generatePolicy(PQ)
4 Efficiency_X ← evaluatePolicy(PX);
5 Go to step 2 if continuing the learning of the Q-learning;
6 Return (PQ,PX);
7 End.
```

Fig. 1. Pseudo code of the performance evaluation algorithm

```
Algorithm EvaluatePolicy(P);
1 Initiating Correct=0, Wrong=0, CostP=0, CostA*=0;
2 For each s ∈ S:
       CostP = cost(s, s_goal, P);
       CostA*= cost(s, s_goal, PA*);
       If CostP <= CostA*
           Correct ← Correct+1;
       Else
           Wrong ← Wrong+1;
3 P ← (Correct / (Correct + Wrong)) * 100;
4 Return (P);
5 End.
```

Fig. 2. Pseudo code of the algorithm of evaluation of a policy *P*

Fig. 1 and 2 present the pseudo code of the performance evaluation algorithm which compares the performance of the Q-leaning algorithm [20] and an algorithm *X*

considering any estimation policy. Fig.1 shows that in each iteration is applied an evaluation function that computes the efficiency of the algorithms. Each iteration represents the state transition by applying an action *a* using the current policy. In step 3 the X algorithm is used to generate a new policy derived from *PQ*. Finally, the efficiency of *PX* is computed and the process continues until a stop condition is reached. Fig. 2 presents the algorithm used for measuring the efficiency of a given policy *P*. The *cost* function is used to find the best path between states *s* and s_goal from a given policies

4.1 Using the K-NN Algorithm to Generate Policies

The K-NN algorithm [1] is used to generate the values in the learning table of an agent based on the Q-learning algorithm. The K-NN algorithm receives as input a set of instances produced by a policy which is generated during learning phase. The aim is to generate a new set of values in an attempt to approximate the agent to an optimal policy.

We define as *arrangement* the set of instances generated during a cycle of steps carried out by the Q-learning algorithm. It is interesting to notice that for each state, four instances are generated (one for each action) and they represent the values learned by the agent. Therefore, the number of instances is the same for any policy. Thus, each instance of the arrangement has attributes for representation of the state in form of the expected reward for the actions north, south, west and east; an action and an attribute representing the reward for the action.

These vectors generated by the Q-learning agent during learning will be compared with a similarity function (cosine) and after that, the nearest instances will be found. Eq. 2 describes how the K-NN can be used to generate estimates of the values for the arrangement formed by the training instances.

$$PKNN\ (s,a) \leftarrow \frac{\sum_{i=1}^{K} PQ_i(.,.)}{K} \qquad (2)$$

where *PKNN(s,a)* represents the values of the reward calculated for a given state *s* and an action *a*, *K* represents the number of neighbors used and $PQ_i(.,.)$ represents the 1^{st}-nearest neighbor instance found in training set generated by the policy *PQ*.

4.2 The K-learning Method

The method described previously can generate intermediate policies; however, it is not guaranteed that these policies present a good quality. In some cases, the values of rewards for a given state change closer to a set of correct rewards. On the other hand, states that have high rewards can become less important. This occurs because all the states will have their values of reward modified, even those whose rewards are good.

The *K-learning* is a hybrid learning method resulting from the merge of the Q-learning and K-NN algorithms which attempts to solve the above mentioned problem by selecting always the policy that improves the performance of the agent. The learning

of the Q-learning algorithm is stored in a table of name *PQ(s,a)* and the learning of the K-NN in a table *PK(s,a)*. The best values of these tables obtained during the learning phase are stored in a new table, which is called *PKI(s,a)* (this is the policy generated by K-learning).

This method has been created from experimental observations where the efficiency of the methods varies during the learning period. In many cases, the performance of the K-NN has made the rewards discovered for the Q-learning less interesting, i.e., states that produce correct decision started to produce errors[1]. To solve this problem, the K-learning modifies only rewards of states that lead to learning errors. When one of the methods is superior to the other, the agent switches the learning method. This makes the agent able to find the action policy that maximizes its performance and decreases the amount of necessary steps. Fig. 3 presents the pseudo code of the K-learning method.

```
Algorithm K_learning(PQ,PKI);
1 Initiating PQ=0, PKI=0;
2 For each iteration repeat:
       While Stop_Conditions <> true do
        If evaluatePo licy(PQ) > evaluatePo licy(PKl)
             Then
                  CurrentPol icy ← PQ
             Else
                 CurrentPol icy ← PKl
3 Return (PQ,PKI);
4 End.
```

Fig. 3. Pseudo code of the K-learning method

Fig. 3 shows that the performance of the Q-learning and K-NN algorithms is compared for all iteration. When the performance of one of them is higher, the policy is stored into a new table (*CurrentPolicy*) which represents the current action policy. The results of this hybrid learning method are presented in Section 5.

4.3 The Simulated Environment

To represent the environment of simulation (road mesh), the crossing had been represented as states. The states of the virtual environment have characteristics of situation traffic: free, low congestion, high congestion, blocked and unknown. Each situation has different values that are used to compute the rewards received by the agent. The agent moves taking one of the following actions: forward, backward, right and left. After each move (transition), the agent will start to have a new state generated by the environment and new actions could be taken. The agent will know if its action was positive or negative through the reward granted by the environment. Eq. 3 describes the current state *s* updated by the next state added to one reward *r*.

$$Q(s,a) \leftarrow \partial(s',a') + r \tag{3}$$

[1] The error definition has been discussed in Section 4.

The transition value ∂ is computed by the agent for a reinforcement scale signal. The agent must choose actions that tend to increase the sum of the values of the reinforcement signal r, as time goes by. The agent can learn how to make this through systematic try-and-test while it interacts with the environment. Thus, the agent will know if its action was positive or negative through the change of behavior of the environment (traffic). If the agent finds itself in a congested traffic state and after this its action goes into to the little congested state, it then receives a positive reward, but if the action takes it to a very congested state a negative reward is received.

5 Experiments

Experiments were carried out with the Q-learning algorithm to evaluate its efficiency considering factors such as: the number of iterations necessary for the agent to reach its best efficiency; the quality of reward policy; variations in learning rate; discounting factor and the values of reward for the congestion conditions. In this section we present experiments that allow identifying the importance of each one these factors in the learning process.

The experimental protocol to evaluate the efficiency of the algorithms includes twenty repetitions of each type of experiment using ten different environments sizes. This is required due to the variation in the algorithm efficiency in single environments. This occurs because the values generated during the learning phase are stochastic. The values of efficiency presented in this section represent average values. The best values used as learning rate for the Q-learning algorithm are between 0.10 and 0.20 and the best values for the discount factor are between 0.85 and 0.95.

Fig. 4 presents a comparative among: efficiency, number of states and amount of steps. Fig. 4 shows that for environments smaller than 25 states, about 500 steps are necessary to reach the best efficiency. For environments up to 81 states we observe that it is necessary about 5,000 steps. Environments above 100 states need a more than 20,000 steps to reach the best efficiency. Fig. 4 and 5 show that in large environments the agent needs a high number of cycles due to the need of visiting each state many times to accomplish the learning.

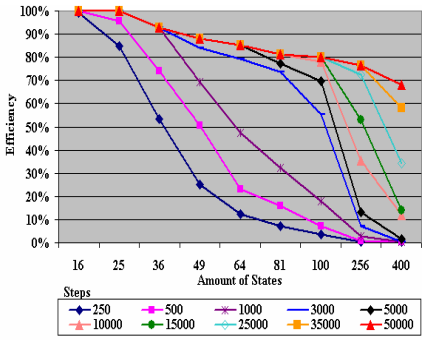

Fig. 4. Learning curves for the Q-learning: efficiency *vs* n. of states *vs* n. of cycles

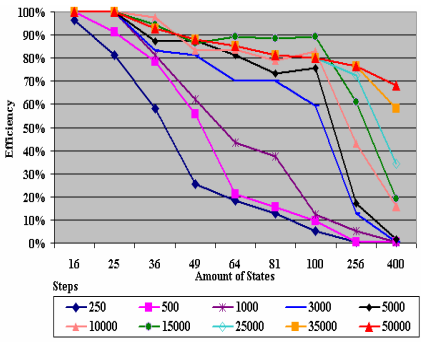

Fig. 5. Leaning curves for the 1-NN: efficiency *vs* n. of states *vs* n. of cycles

The instances generated by the Q-learning and used by K-NN are stored into a K-Table. Further, the policy represented in the K-Table is evaluated using the algorithm *A** as discussed in section 4. The experiments with the K-NN algorithm have taken place in environments with size up to 400 states and with up to 50,000 cycles. The average efficiency of the K-NN algorithm was evaluated considering K=1 because this was the value that has presented the best results.

It is possible to observe that the K-NN algorithm presents a better performance than Q-learning for most of the environment sizes (Fig. 4 and 5). This fact motivates us to propose the hybrid learning method, namely K-learning.

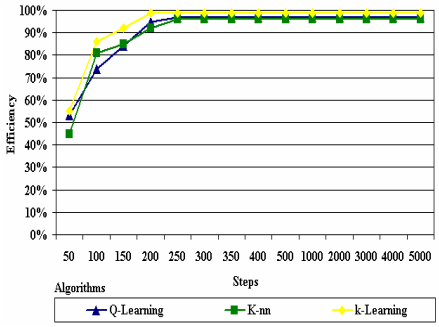

Fig. 6. Efficiency of the Q-learning, 1-NN and K-learning algorithms for a 16-state environment

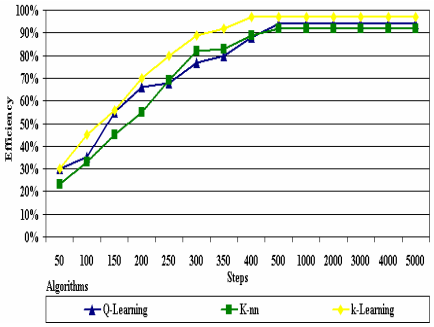

Fig. 7. Efficiency of the Q-learning, 1-NN and K-learning algorithms for a 25-state environment

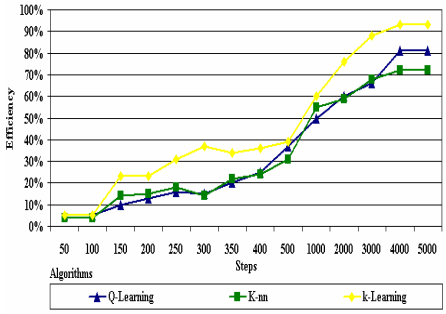

Fig. 8. Efficiency of the Q-learning, 1-NN and K-learning algorithms for a 64-state environment

5.1 K-learning versus Q-learning and K-NN

To evaluate the performance of the K-leaning algorithm we have carried out experiments in environments with size between 16 and 64 states. Fig. 6 to 8 show the efficiency curves for the K-learning hybrid method, the Q-learning and the K-NN algorithms. The curves demonstrate that, in general, the K-learning hybrid method presents a better efficiency than Q-learning and K-NN algorithms in any learning cycle. We can also observe that the number of steps necessary to find a good action

policy decreases significantly relative to the Q-learning and K-NN algorithms. In 16-state environments there is an average reduction of 18% in the number of steps to achieve the best efficiency. For 25-state environments the reduction is 20%. Finally, for 64-state environments the K-learning method has achieved the best efficiency with 12% less steps. The values of the Tab. 1 present the superiority of the K-learning method to the Q-learning and K-NN algorithms. It is possible to observe that the K-learning method has a higher average performance relative to the other methods. It also requires a lower number of iterations to find a good action policy. This shows that this method is able to adapt itself to different environments. The good average performance results from the policy generation technique gets the values learned by the agent closer to the values of an optimal action policy.

Table 1. Average superiority of the K-learning method to the Q-learning and K-NN algorithms

Environment Size (number of states)	Q-learning (%)	K-NN (%)
16	2	3
25	3	5
64	12	21

6 Conclusion and Discussion

This paper presented a novel hybrid learning method that is more efficient that conventional learning algorithm such as the Q-learning, K-NN algorithms. The experiments carried out on different environment sizes have show that even having a higher computational cost, the K-NN algorithm achieves satisfactory results since it generally finds superior or equivalent solutions to Q-learning with a low number of iterations. The hybrid K-learning method has integrated the robustness of the Q-learning with the efficiency of the K-NN algorithm to modify the learning values. This hybrid learning method has succeeded to optimize the agent learning in terms of efficiency and the number of iterations necessary to achieve a good performance.
In this paper we have also presented a novel evaluation methodology which is robust in dealing with partially known and complex environments. Such an evaluation methodology also allows the setup of suitable learning parameters for RL algorithms. This is an important point because these factors have great relevance in the performance of a learning agent.

Even though the results are encouraging, additional experiments are necessary to answer some open questions, such as: (i) the comparison of different forms of updating the current policy. The update of the current policy could be partial rather than global (only with the best reward values); (ii) a heuristic function could be used to speedup the RL [3]; (iii) A multi-agent system could be used for exploring distant regions of the target state where rewards are low. Multi-agent reinforcement learning is a very active subject of research [2] [16]; (iv) another question consists in evaluating the algorithms in dynamic and noisy environments. Since the K-learning uses results from different algorithms, it should be robust in situations where the reward values vary since is well

known that the K-NN is robust to deal with noise in training data; however, this hypothesis will be verified in future research.

7 References

1. Aha, D.W., Kibler, D., Albert, M.K. Instance-based Learning Algorithms, Machine Learning. 6(1), pp. 37-66, 1991.
2. Almeida, A., Ramalho, G. L., Santana, H. P., Tedesco, P., Menezes, T. R., Corruble, V., Chevaleyre, Y. (2004). Recent Advances on Multi-Agent Patrolling. In Advances in Artificial Intelligence – SBIA 2004. LNAI 3171, pp. 474-483. Berlin: Springer
3. Bianchi, R. A. C., Ribeiro, C. H. C., Costa, A. H. R. Heuristically Accelerated Q-learning: A New Approach to Speed Up Reinforcement Learning. XVII SBIA'04: pp: 245-254, 2004.
4. Downing, K. L. Reinforced Genetic Programming. Genetic Programming and Evolvable Machines, 2(3):259-288, 2001.
5. Ernst, D., Geurts, P., Wehenke, L. Tree-Based Batch Mode Reinforcement learning. Journal of Machine Learning Research, April 2005, Volume 6, pp 503-556
6. Figueiredo, K., Vellasco, M., Pacheco, M., Souza, M., Reinforcement Learning Hierarchical Neuro-Fuzzy Politree Model for Control of Autonomous Agents," his, pp. 130-135, Fourth Int. Conference on Hybrid Intelligent Systems (HIS'04), 2004.
7. Henderson, J., Lemon, O., Georgila, K.. Hybrid reinforcement/supervised learning for dialogue policies from COMMUNICATOR data. In Proc. IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems, Edinburgh, 2005.
8. Kaelbling, L. P., Littman, M. L, Moore, A. W., Reinforcement Learning: A survey. Journal of Artificial Intelligence Research. Vol. 4, pp. 237-285, 1996.
9. Levner, I., Bulitko,V., Madani, O. , Greiner, R.. Performance of Lookahead Control Policies in the Face of Abstractions and Approximations. Technical Report. Publisher: Springer-Verlag., Vol. 2371/2002. pp. 299-307.
10. Maes, P. Artificial Life Meets Entertainment: Lifelike Autonomous Agents, Communications of ACM, Vol. 38, No. 11, pp. 108-114, 1995.
11. Mitchell, T. Machine Learning. Boston:McGraw-Hill, 1997.
12. Ramon, J. On the convergence of reinforcement learning using a decision tree learner. Proceedings of ICML-2005 Workshop on Rich Representation for Reinforcement Learning, Bonn, Germany, 2005.
13. Ribeiro, C. H. C. A Tutorial on Reinforcement Learning Techniques, Int. Joint Conference on Neuronal Networks, Washington: INNS Press, 1999.
14. Russel, S., Norvig, P., Inteligência Artificial, 2 ed., Rio de Janeiro: Editora Elsevier, 2004.
15. Ryan, M. R. K. Hierarchical Reinforcement Learning: A Hybrid Approach. PhD Thesis, University of New South Wales, School of Computer Science and Engineering, 2004.
16. Santana, H., Ramalho, G., Corruble, V., Ratitch, B. Multi-Agent Patrolling with Reinforcement Learning In proc. 3rd International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'04). pp. 1122-1129. New York: ACM
17. Siedlecki, W., Sklansky, J., A note on Genetic Algorithms for Large-Scale Selection, Pattern Recognition Letters, Vol. 10, pp. 335-347, 1989.
18. Sutton, R.S., Barto, A.G. Reinforcement Learning: An Introduction. Cambridge, MA. MIT Press, 1998.
19. Tesauro, G. Temporal Difference Learning and TD-Gammon. Communications of the ACM, Vol. 38, No. 3, pp. 58-68, 1995.
20. Watkins, C. J. C. H., Dayan, P. Q-learning, Machine Learning, 8 ed., pp. 279-292, 1992.