2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)

Reinforcement Learning Algorithms in Global Path Planning for Mobile Robot

Valentyn N. Sichkar
Department of Control Systems and Robotics
ITMO University
Saint-Petersburg, Russia
vsichkar@corp.ifmo.ru

***Abstract*—The paper is devoted to the research of two approaches for global path planning for mobile robots, based on Q-Learning and Sarsa algorithms. The study has been done with different adjustments of two algorithms that made it possible to learn faster. The implementation of two Reinforcement Learning algorithms showed differences in learning time and the methods of building path to avoid obstacles and to reach a destination point. The analysis of obtained results made it possible to select optimal parameters of the considered algorithms for the tested environments. Experiments were performed in virtual environments where algorithms learned which steps to choose in order to get a maximum payoff and reach the goal avoiding obstacles.**

***Keywords*—reinforcement learning, Q-Learning algorithm, Sarsa algorithm, path planning, mobile agent**

I. Introduction

Reinforcement Learning represents a class of tasks in which mobile robot (in this study considered as mobile agent), acting in a particular environment, must find an optimal strategy for interaction with it. One of the popular methods used to solve such problems is Q-Learning and Sarsa algorithms. For training mobile agent information is provided in form of reward that has a certain quantitative value for each transition of the mobile agent from one state to another (from one point to another). No other additional information is provided to train the agent. The most important feature of Q-Learning and Sarsa algorithms is that they can be used even when the mobile agent does not have prior knowledge of the environment.

When Reinforcement Learning algorithms are working, the state-action pairs estimation function is being constructed. In the standard view, this function is displayed as a table whose inputs are these state-action pairs. One of the conditions for convergence of the algorithm in the case of using a table representation of the function is multiple testing of all possible state-action pairs to find the optimal path in a virtual environment with obstacles. The goal of the mobile agent is to find a behaviour policy that maximizes the expected amount of reward. Algorithms show the ability of Reinforcement Learning when the mobile agent does not know anything about the environment and learns the optimal behaviour in which the reward for actions is maximum, and the reward is awarded not immediately, for some action, but for the sequence of actions. This is what this study is devoted to, based on the algorithm of Q-Learning and its modification of Sarsa.

II. Algorithm Q-Learning And Its Modification Sarsa

The task of Reinforcement Learning in general form is formulated as follows. For each transition of the mobile agent from one state to another, the scalar value is assigned, called an award. The agent receives the award for making the transition. The goal is to find actions that maximize the expected reward amount.

In order to accomplish this goal Q-Learning algorithm uses Q-function, an argument of which is the action performed by the agent [1]. This allows an iterative way to build Q-function and thereby find the optimal control policy. The expression for updating Q-function is as following:

$$Q(x_t, a_t) \leftarrow r_t + \gamma \cdot \max Q(x_{t+1}, a_t) \qquad (1)$$

where r_t is the reward received when system moves from the x_t state to the x_{t+1} state, γ is discount factor which lies in the range from 0 to 1, and a_t is the action selected at time t from set of all possible actions.

Q-value estimates are stored in the two-dimensional table whose inputs are state and action. Equation (1) is usually combined with a time difference method [2]. With the parameter of time difference method equal to zero, only the current and subsequent values of prediction of Q-values are involved in the update. In this case, the method is called one-step Q-Learning. The expression for one-step Q-Learning is as following:

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \lambda \cdot (r_t + \gamma \cdot \max Q(x_{t+1}, a_t) - Q(x_t, a_t)) \qquad (2)$$

By analysing equation (1) it can be concluded that using maximum to estimate the next action is not the best solution. In the early stages of learning, Q-value table contains estimates that are far from ideal, and even in the later stages, using maximum can lead to a re-evaluation of Q-values. In addition, the rule for updating the Q-Learning algorithm in combination with the time difference algorithm requires zero-parameter value when choosing actions based on non-greedy policies. In this case, a non-greedy policy is a policy in which actions are selected with a certain probability, depending on the value of

Q-functions for a given state, unlike a greedy policy, when actions with the highest Q-value are selected. These disadvantages caused modification of the Q-Learning algorithm, which is called SARSA (State-Action-Reward-State-Action). The main difference between this algorithm and the classical one is that the max operator is removed from the Q-value update rule. As a result, it is guaranteed that the time difference error will be calculated correctly, regardless of whether actions are chosen according to the greedy policy or not.

III. Q-Value Table Approximation Methods

One of the easiest ways to efficiently work with a large dimension of state space is discretization. At discretization, the space of states is divided into regions of small size; each such region is input to table of Q-values. By using this approach, an approximation of states is obtained. Success in this case directly depends on how well this partition allows us to represent the function of Q-values. On the one hand, for greater accuracy, it is required to divide space into smaller areas and, as a result, to use larger Q-value table, which will result in the need for more updates during training. On the other hand, splitting into larger areas may lead to the impossibility of achieving optimal control policy. The described method had a successful application in work for the task of balancing a cart-pole [3], which in the field of reinforcement training is considered to be a classic example.

There are also methods to speed up the learning process when using large Q-value tables. One of these methods is the Hamming distance method [4]. When using this method, all states are represented in binary form, and the similarity threshold is set, namely the number of bits by which one state may differ from another. When Q-values are corrected, updating is simultaneously performed both for selected state and for all states to which the Hamming distance from the selected one is less than the specified threshold. Consequently, the spread of Q-values in the table is accelerated.

Another method called CMAC (Cerebellar Model Articulator Controller) is a compromise between using simple Q-value table and continuous approximation of the function [5]. CMAC approximation structure consists of several layers. Each layer is divided into intervals of the same length (called as tiles) using a quantizing function. Since each layer has its own quantization function, the tiles of the layers are shifted relative to each other. Consequently, the state of the system applied to inputs of the CMAC is matched with a set of overlapping offset tiles. However, despite successful application, this algorithm requires fairly complex settings. The accuracy of the approximated function is limited by the resolution of quantization. High quantization accuracy requires more weights and longer study of the environment.

RBF networks (Radial Bases Functions) are closely related to CMAC and simple tables [6]. When using this method of approximation, instead of Q-values table, a grid of Gauss functions or quadratic functions is stored. State of the system is passed through all functions, after which values of the functions are summed, and the result is approximated value.

There is also such a method of approximating Q-value table as statistical cluster analysis [7]. When using this method, each action is associated with a multitude of clusters that represent evaluations of actions in a particular class of situations. During updating, Q-values for current state are updated for all states belonging to this cluster. However, there are following limitations of this method: the difficulty of setting parameters for formation of semantically significant clusters and the fact that cluster formed once cannot be broken later.

It is known that multilayer perceptron is also good approximator of functions. There is Kolmogorov theorem on the mapping of neural networks (Kolmogorov mapping neural network existence theorem), which proves that neural networks of direct distribution with three layers (input layer, hidden layer, output layer) can accurately represent any continuous function [8]. The use of neural networks to approximate Q-function has following advantages: effective scaling for large dimension input space, generalization for large and continuous state spaces, the possibility of implementation on parallel hardware.

IV. Building a Path By Mobile Agent

The effectiveness of the algorithms described in this paper was analysed using developed software simulator of the mobile agent operating in the 2-dimensional virtual environment. The agent was tasked to achieve the goal, avoiding collisions with obstacles. The virtual environment is divided into cells and obstacles are occupied some of these cells. If the mobile agent falls into one, it counts as collision. Example of the environment in which experiments were conducted is presented in Figure 1.

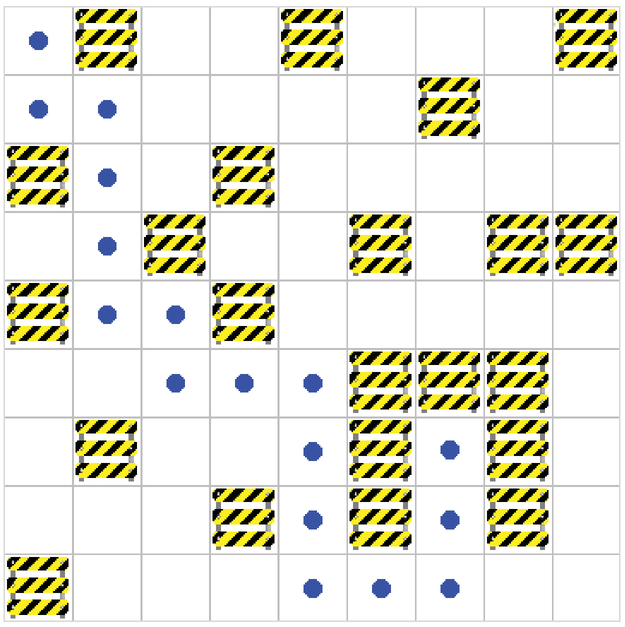

Fig. 1. Virtual environment with obstacles and found path

The initial position of the mobile agent is in the upper left corner. Figure 1 shows the path found after training. At each step of operation, the agent could choose one of four possible actions: moving forward, moving backwards, moving left, moving right. Exceptions are the boundary and angular positions, where one or more options to act are missing. At the same time, in one action the mobile agent can take only one step in one of the chosen direction.

The Q-value table is represented by the possible actions of the mobile agent and is filled with rewards according to the chosen behaviour. In Reinforcement Learning algorithms, processes of exploration and exploitation play an important role. At the first stage, it is necessary to investigate the environment as best as possible by choosing lower priority actions. At the final stages, it is necessary to proceed directly to the operation, choosing more priority actions. A smooth transition between exploration and exploitation can be described using Boltzmann distribution, which has the following form:

$$P(a_t \mid x_t) = [\exp(Q(x_t, a_t) / T)] / [\Sigma\exp(Q(x_t, a_t) / T)] \quad (3)$$

where T is the temperature that controls the degree of randomness of the choice of action with the highest Q-value.

V. Experimental Results

Experiments were done in the virtual environment with obstacles and the mobile agent was as the object that moved in this virtual environment, studying it in order to find the path to the target point. Program code was written in Python 3 using libraries to visualize moving process of mobile agent as the object in the environment. At the initial stage of training, visualization of agent transitions from one point to another was especially slowed down that it could be possible to see and to study its behaviour and adjust the parameters.

Experiments were conducted for Q-Learning algorithm and its modifications Sarsa with different parameters. A total of 50 experiments were conducted. It was necessary to pay special attention to the range of temperature variation T and its rate of change since the convergence of the algorithm strongly depends on these parameters. Experimentally was established that the optimal solution between quality and learning speed is the use of interval for T from 0.01 to 0.04 for 1000 stages. Final table with Q-values for Q-Learning algorithm is shown in Table I.

Q-values of the filled table shows which final actions were chosen by the agent after studying the environment (the values are highlighted in grey). The sequence of final actions to achieve the goal after Q-table is filled with knowledge is as following: down-right-down-down-down-right-down-right-down-right-down-down-right-right-up-up. In the course of the experiment with the Q-Learning algorithm, the shortest path for achieving the goal consists of 16 steps, and the longest path to the goal is 185 steps. Figure 2 and Figure 3 shows charts with learning process for Q-Learning algorithm.

Charts show the number of episodes over the number of steps (Figure 2) and the number of episodes over cost for each episode (Figure 3). From the charts, it can be seen that starting from about 300th episode, the mobile agent found the path to the target, and its award for committed actions grows.

TABLE I. Q-Values For Q-Learning Algorithm With Found Best Actions

Up	Down	Right	Left
1.167617e-05	7.172375e-04	-3.701764e-01	9.011239e-06
0.000028	-0.222179	1.712710e-03	0.000015
0.000054	0.003944	7.389733e-07	0.000067
0.000099	0.008411	2.200491e-06	0.000068
0.000105	0.017174	-2.822695e-01	0.000549
0.000071	0.000034	1.853133e-02	0.000451
-0.206386	0.037152	-2.221786e-02	0.000222
0.001505	0.000234	6.961416e-02	0.000018
-0.267697	0.125094	7.383372e-04	0.001612
0.000422	-0.252828	2.666055e-01	0.011204
0.000012	0.379808	-1.485422e-01	0.013184
0.009001	0.510842	-2.063857e-01	-0.182093
0.027898	0.040092	6.458811e-01	0.001964
-0.104662	0.060179	7.727067e-01	0.015624
0.888066	0.061701	8.728341e-03	0.093561
0.997643	0.093159	-1.570568e-01	-0.206386

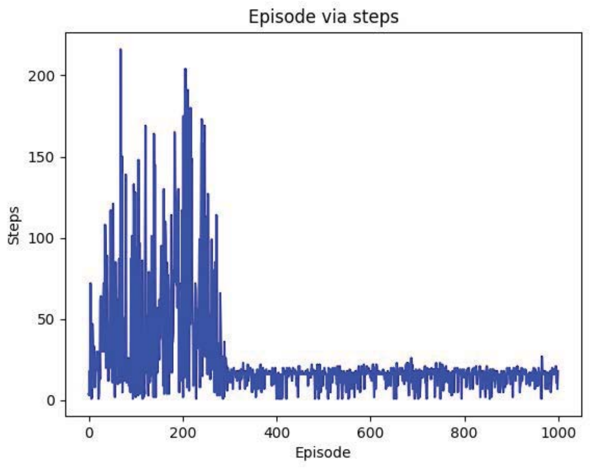

Fig. 2. Episode via steps for Q-Learning algorithm

Q-Learning algorithm showed the best result with following parameters: λ = 0.5 (learning rate), γ = 0.99 (discount factor). Table II displays the final comparison of Q-Learning and Sarsa algorithms.

Experiments for comparison of Q-Learning and Sarsa were also done in another virtual environment in order to visualize the difference in the behaviour of the mobile agent. The initial position of the mobile agent was also located in the upper left corner, and occupied cells were represented as a cliff. The goal of the agent is to reach the top right corner and not fall into the cliff. The paths found by the mobile agent for Q-Learning algorithm and for Sarsa are shown in Figure 4.

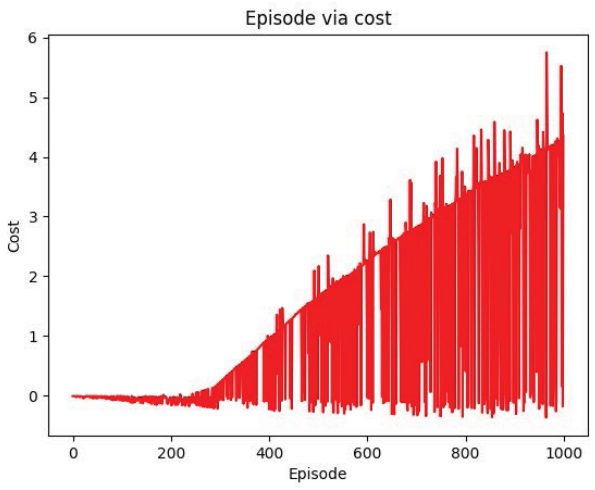

Fig. 3. Episode via cost for Q-Learning algorithm

TABLE II. COMPARISON OF Q-LEARNING AND SARSA ALGORITHMS

Algorithm	**Parameters**			
	Learning rate	***Discount factor***	***Minimum steps***	***Maximum steps***
Q-Learning	0.5	0.99	16	185
Sarsa	0.5	0.99	19	235

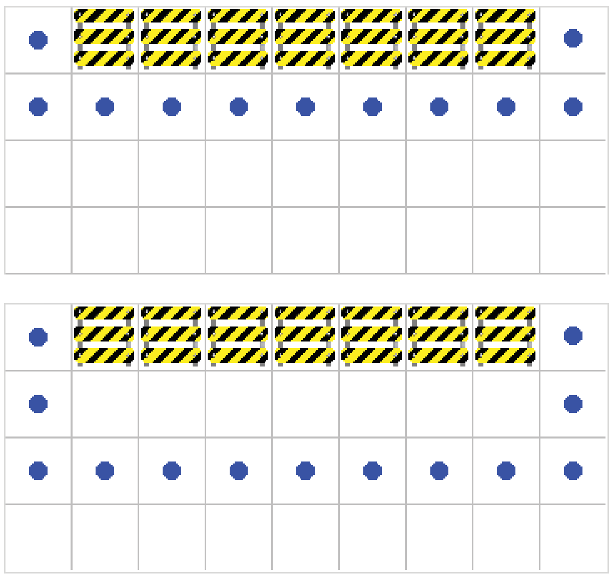

Fig. 4. Comparison analysis of Q-Learning and Sarsa algorithms

Figure 4 shows that when using Q-Learning algorithm (in the upper part), the mobile agent tries to minimize the number of necessary actions and achieve the goal as quickly as possible. However, in this case, the risk of falling off the cliff increases. In the case of using Sarsa algorithm (in the bottom part), the agent gives priority to security and finds the optimal safe distance to the cliff to minimize the risk of falling. However, in this case, the number of necessary actions to achieve the goal increases.

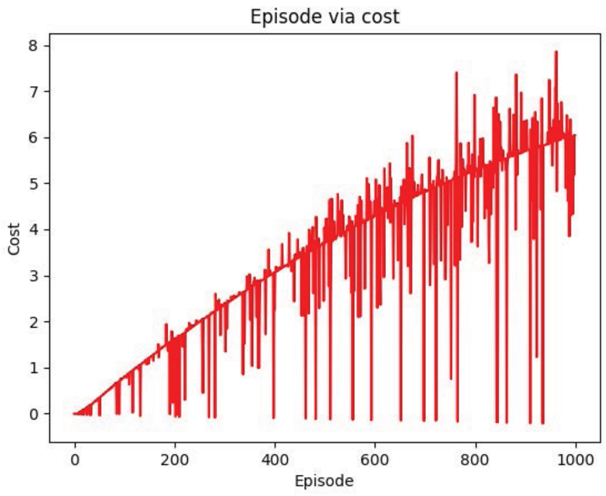

Fig. 5. Q-Learning algorithm learns to avoid falling from the cliff

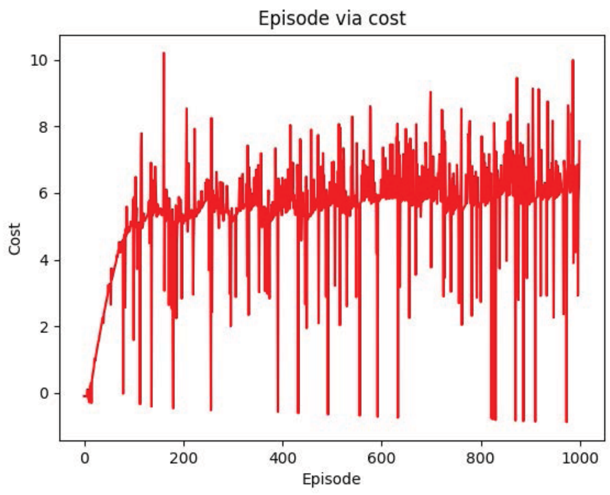

Fig. 6. Sarsa algorithm learns to avoid falling from the cliff

Charts in Figure 5 and Figure 6 also show the difference in the performance of Q-Learning and Sarsa algorithms learning how to avoid falling from the cliff. It can be seen that Sarsa algorithm needs more cost during learning in comparison of Q-Learning algorithm. And it means that Sarsa takes more steps to reach the goal than Q-Learning.

VI. CONCLUSIONS

This paper studies Q-Learning algorithm and its modifications of Sarsa algorithm for the tasks of finding the path in given environment. Experiments with these algorithms were conducted with software simulator of the agent operating in the virtual environment with obstacles. During experiments, the optimal parameters of the algorithms were found, and their efficiency was compared. Algorithms showed the fastest convergence with the parameter of learning rate $\lambda = 0.5$ and with the parameter of discount factor $\gamma = 0.99$. Q-Learning showed faster convergence than its Sarsa modification. However, with Sarsa algorithm the agent moved along the safer trajectory, which was shown by additional experiments on another virtual environment with simulation of the cliff. Consequently, each of the investigated algorithms has its own advantages in speed (Q-Learning) and in safety (Sarsa), which makes them possible to use in solving certain types of tasks.

REFERENCES

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, “Deep Reinforcement Learning: A Brief Survey,” IEEE Signal Processing Magazine, vol. 34, pp. 26-38, 2017.

[2] R. Sutton, S. Richard, “Learning to predict by the methods of temporal differences,” Machine Learning, vol. 3, pp. 9-44, 1988.

[3] S. Nagendra, N. Podila, R. Ugarakhod, K. George, “Comparison of reinforcement learning algorithms applied to the cart-pole problem,” 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 26-32, 2017.

[4] H. Jegou, M. Douze, C. Schmid, “Product quantization for nearest neighbor search,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, pp. 117–128, 2011.

[5] L. Kurtaj, V. Shatri, I. Limani, “On-line learning of robot inverse dynamics with cerebellar model controller in feedforward configuration,” International Journal of Mechanical Engineering and Technology, vol. 9, pp. 445-460, 2018.

[6] P. Roy, S. Adhikari, “Radial Basis Function based Self-Organizing Map Model,” IOSR Journal of Engineering, vol. 8, pp. 46-52, 2018.

[7] B. Everitt, S. Landau S., M. Leese and D. Stahl, “Cluster Analysis,” Wiley, 5th edn., 2011.

[8] B. Igelnik, N. Parikh, “Kolmogorov's spline network,” IEEE Transactions on Neural Networks, vol. 14-4, pp. 725-33, 2003.

[9] H. V. Hasselt, A. Guez, D. Silver, “Deep reinforcement learning with double Q-learning,” AAAI Conference on Artificial Intelligence, pp. 2094–2100, 2016.

[10] E. Even-Dar, Y. Mansour, “Learning rates for Q-learning,” Journal of Machine Learning Research, vol. 5, pp. 1–25, 2003.

[11] H. Iima, Y. Kuroe, “Swarm reinforcement learning algorithms based on Sarsa method,” SICE Annual Conference, Tokyo, 2008, pp. 2045-2049.

[12] A. Edwards, W. M. Pottenger, “Higher order Q-Learning,” 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 128-134, 2011.

[13] D. Xu, Y. Fang, Z. Zhang, Y. Meng, “Path Planning Method Combining Depth Learning and Sarsa Algorithm,” 2017 10th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, pp. 77-82, 2017.

[14] F. Tavakoli, V. Derhami, A. Kamalinejad, “Control of humanoid robot walking by Fuzzy Sarsa Learning,” 2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM), Tehran, pp. 234-239, 2015.

[15] A. Habib, M. I. Khan, J. Uddin, “Optimal route selection in complex multi-stage supply chain networks using SARSA(λ),” 2016 19th International Conference on Computer and Information Technology (ICCIT), Dhaka, pp. 170-175, 2016.

[16] R. Lowe, T. Ziemke, “Exploring the relationship of reward and punishment in reinforcement learning,” 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), Singapore, pp. 140-147, 2013.

[17] R. Zhang, P. Tang, Y. Su, X. Li, G. Yang, C. Shi, “An adaptive obstacle avoidance algorithm for unmanned surface vehicle in complicated marine environments,” IEEE/CAA Journal of Automatica Sinica, vol. 1, pp. 385-396, 2014.

[18] R. Ozakar, B. Ozyer, “Ball-cradling using reinforcement algorithms,” 2016 National Conference on Electrical, Electronics and Biomedical Engineering (ELECO), pp. 135-141, 2016.

[19] W. Sause, “Coordinated Reinforcement Learning Agents in a Multi-agent Virtual Environment,” 2013 12th International Conference on Machine Learning and Applications, pp. 227-230, 2013.

[20] D. A. Vidhate, P. Kulkarni, “Enhanced Cooperative Multi-agent Learning Algorithms (ECMLA) using Reinforcement Learning,” 2016 International Conference on Computing, Analytics and Security Trends (CAST), pp. 556-561, 2016.

[21] B. N. Araabi, S. Mastoureshgh, M. N. Ahmadabadi, “A Study on Expertise of Agents and Its Effects on Cooperative Q-Learning,” IEEE Transactions on Evolutionary Computation, vol. 14, pp. 23-57, 2010.

[22] A. Deepak, P. Kulkarni, “New Approach for Advanced Cooperative Learning Algorithms using RL methods (ACLA),” VisionNet’16 Proceedings of the Third International Symposium on Computer Vision and the Internet, 2016.

[23] M. Fairbank, E. Alonso, “The divergence of reinforcement learning algorithms with value-iteration and function approximation,” 2012 International Joint Conference on Neural Networks (IJCNN), 2012, pp. 1-8, 2012.

[24] Y. Yuequan, J. Lu, C. Zhiqiang, T. Hongru, X. Yang, N. Chunbo, “A survey of reinforcement learning research and its application for multi-robot systems,” Proceedings of the 31st Chinese Control Conference, pp. 3068-3074, 2012.

[25] W. Xu, J. Huang, Y. Wang, C. Tao, X. Gao, “Research of reinforcement learning based share control of walking-aid robot,” Proceedings of the 32nd Chinese Control Conference, pp. 5883-5888, 2013.

[26] M. van der Ree, M. Wiering, “Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play,” 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), pp. 108-115, 2013.

[27] Zhi-Xiong Xu, Xi-Liang Chen, Lei Cao, Chen-Xi Li, “A study of count-based exploration and bonus for reinforcement learning,” 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 425-429, 2017.

[28] S. Wender, I. Watson, “Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar,” 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 402-408, 2012.

[29] N. Chauhan, N. Choudhary, K. George, “A comparison of reinforcement learning based approaches to appliance scheduling,” 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), pp. 253-258, 2016.

[30] F. Cardenoso Fernandez, W. Caarls, “Parameters Tuning and Optimization for Reinforcement Learning Algorithms Using Evolutionary Computing,” 2018 International Conference on Information Systems and Computer Science (INCISCOS), pp. 301-305, 2018.

[31] W. Lu, J. Yang, H. Chu, “Playing Mastermind Game by Using Reinforcement Learning,” 2017 First IEEE International Conference on Robotic Computing (IRC), pp. 418-421, 2017.

[32] M. D. Kaba, M. G. Uzunbas, S. N. Lim, “A Reinforcement Learning Approach to the View Planning Problem,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5094-5102, 2017.

[33] H. Cetin, A. Durdu, “Path planning of mobile robots with Q-learning,” 22nd Signal Processing Conference, pp. 2162-2165, 2014.