Implementation of Q - Learning algorithm for solving maze problem

D. Osmanković, S. Konjicija
Department of Automatic Control and Electronics
University in Sarajevo, Faculty of Electrical Engineering in Sarajevo
Zmaja od Bosne bb, 71000 Sarajevo, Bosnia and Herzegovina
Phone: (387) 61 307 023 E-mail: dinko.osmankovic@etf.unsa.ba, samim.konjicija@etf.unsa.ba

Abstract - Machine learning is very important in several fields ranging from control systems to data mining. This paper presents Q - Learning implementation for abstract graph models with maze solving (finding the trajectory out of the maze) taken as example of graph problem. The paper consists of conversion of maze matrix to Q - Learning reward matrix, and also the implementation of Q - Learning algorithm for the reward matrix (similar to minimizing criteria matrix in dynamic programming). This implementation is on higher level of abstraction, so other representations can be used (artificial neural networks, tree etc.). For the testing of Q - Learning algorithm, maze solving problem was visualized in MATLAB programming language with the found trajectory marked on the maze. The maze in this paper is defined with starting position in the top left corner and the exit in the bottom right corner. The performance of the algorithm is measured for different scales of the problem.

Keywords – Artificial intelligence, reinforcement learning, maze, machine intelligence, agent

I. INTRODUCTION

MACHINE learning is a field of artificial intelligence which deals with algorithm researching that solves problems based on collected empirical data called experience. In the case of robot being an intelligent agent, this experience is collected from sensors (sonar, laser, camera etc.)

This field of artificial intelligence is very broad since it contains knowledge from several areas in science: control theory, theoretical computer science, data mining, statistics, computer vision, philosophy, psychology etc. [1]

Several problems were solved using machine learning techniques including speech recognition [2] and backgammon AI player called TD-Gammon [3].

Reinforcement learning is one of several machine learning techniques of artificial intelligence. The basic principle comes from action – state principle of agent – environment system [1]. Every action agent makes on environment, it is rewarded and based on this reward it is possible to model almost any problem on abstract level to be solved using reinforcement learning technique [1].

Reinforcement learning, particularly, has been used to solve variety of problems: multi – robot cooperation [4], AI game play in PC games [5], robotic manipulator control [6], and soccer robot teams [7].

The motivation for this work comes from the problem of modelling typical problems so it fits the definition of reinforcement learning problem. Typical problem is graph traversing, particularly maze solving. For this paper we selected Q Learning algorithm to solve this problem which was implemented in MATLAB.

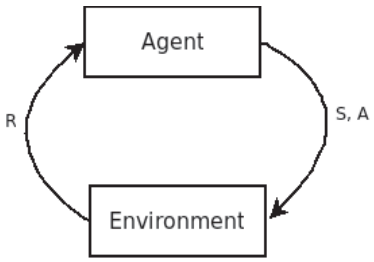

Fig. 1. Agent – environment interaction model diagram

First, we defined some rules for the maze: starting point is top – left corner, and exit is bottom – right corner. Intelligent agent can move in 4 directions: north, west, south and east. As it moves towards the exit, the reward increases, and the algorithm terminates with the agent sitting in exit point.

This paper is a proof-of-concept to show how can abstract implementation of Q – Learning be used in maze solving. In section II mathematical model of maze problem will be presented. In the next section the particular implementation will be explained and in the last section performance of the implemented algorithm will be analyzed.

II. MATHEMATICAL MODELLING OF THE MAZE PROBLEM

First, we need to give some abstract definitions of the problem. This is a problem of finding the best interaction between the agent and the environment where it operates.

Diagram showing principal interaction between the agent and the environment is given in Fig. 1. For the given state S, agent makes action A, and the environment responds with scalar reward R. If we model the problem in such way so the agent makes set of actions A_i (where $i \in (1, n)$, and n is a finite number from $\mathbb{N}$) and environment responds with minimal (or maximal)

cumulative reward, then this problem is seen as an optimization problem.

For every given time t, the agent needs to recognize its current state $s_t \in S$, and the set of possible actions $A(s_t)$. Agent then chooses action $a \in A(s_t)$, changes its state to s_{t+1}, and gets the reward r_t from the environment. Based on these interactions, the agent needs to learn the mapping given by (1).

$$\pi: S \to A \tag{1}$$

The function given by (1) needs to maximize the sum of rewards received for every action taken. This cumulative reward is given by (2) for Markov decision process with finite state, and by (3) for Markov decision process with no finite state.

$$R = r_0 + r_1 + r_2 + \cdots + r_n. \tag{2}$$

$$R = \sum_t \gamma^t \cdot r_t. \tag{3}$$

where:

γ – discount factor for some future reward, $0 \leq \gamma < 1$

All the equations (1-3) can be found in [1].

Discount factor of value 1 or more is proved to make Q – values diverge [8]. This factor determines if the agent will look for long – term high reward (as the factor approaches value 1) or it will only consider rewards moving it from current state.

Q – learning algorithm takes the same action – state principle, with a difference, in this case, that we write π function as Q.

For the maze problem, agent follows the trajectory that maximizes the reward received making actions to follow that particular trajectory. If the agent falls off the optimal trajectory it receives negative reward or penalty.

Representation of knowledge in Q – learning is a problem for itself. Most of the implementations use table or matrix representations, but it's not a viable approach for large – scale and complex problems. Modern approach is to use artificial neural networks as demonstrated in [3]. For this particular problem, using matrix representation is sufficient.

First step in the Q – learning algorithm is to initialize the values of elements in knowledge matrix which represents state transition matrix. For example, let the maze matrix be matrix A in (4). In this matrix, 1 represents empty space while 0 represents maze walls.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3}$$

One possible solution in writing matrix of initial Q values is to label all possible states with numbers from $[1, n]$ where $n \in \mathbb{N}$ is a number of possible states as written in (4). Note that x labels forbidden state (i.e. maze wall as agent cannot move towards the wall).

$$StateLabels = \begin{bmatrix} x & x & x & x & x \\ x & 1 & 2 & 3 & x \\ x & 4 & x & 5 & x \\ x & 6 & x & 7 & x \\ x & x & x & x & x \end{bmatrix}. \tag{4}$$

To initialize Q values, we create matrix of size n× n where n is number of possible states the agent can move to. If there exits direct transition between states (for the set of actions agent can make) we give value 0, otherwise $-\infty$, for example, there is a direct transition between states 1 and 2 so the Q – value is $Q(1,2) = 0$. Also, there is no direct transition between states 1 and 5, so the Q – value is $Q(1,5) = -\infty$. We get a matrix as written in (5).

$$Q = \begin{bmatrix} -\infty & 0 & -\infty & 0 & -\infty & -\infty & -\infty \\ 0 & -\infty & 0 & -\infty & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & -\infty & 0 & -\infty & -\infty \\ 0 & -\infty & -\infty & -\infty & -\infty & 0 & -\infty \\ -\infty & -\infty & 0 & -\infty & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \end{bmatrix}. \tag{5}$$

Updating values in Q – values matrix is the next step in this algorithm. For, every pair (s_t, a_t) we update the value of $Q(s_t, a_t)$ with (6).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)\big(1 - \alpha_t(s_t, a_t)\big) + \alpha_t(s_t, a_t) \times \Big[r_t + \gamma_t \max_a Q(s_{t+1}, a)\Big]. \tag{6}$$

where:

r_t – reward in time t,
α_t – learning rate in time t, $0 \leq \alpha \leq 1$

Learning rate is a very interesting parameter in Q – learning method. It determines the rate of which the newly acquired knowledge will override the older knowledge. Value 0 indicates that the agent doesn't learn anything, while value 1 indicates that doesn't remember anything and only the recently acquired knowledge will be taken into consideration [9].

One must notice that it is not possible to update Q – values using (6) for s_{t+1} being the final state since it is meaningless to calculate $Q(s_{t+1}, a)$. For that, we need to use (7). Value $Q(s_{t+1}, a)$ corresponds to gain in state s_t when this state is final, its gain is reward received for moving to that state (last received reward in the epoch).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)\big(1 - \alpha_t(s_t, a_t)\big) + \alpha_t(s_t, a_t) \times [r_t + \gamma_t\, r_{t+1}]. \tag{7}$$

Equations (6) and (7) can be found in [1].

Pseudo – code for this algorithm is given in Fig. I.

For each (s, a) initialize matrix of $Q(s, a)$ to initial value;

Observe the current state s;

Do forever (or until final state is reached):

- Select action a and execute it;
- Receive immediate reward;
- Observe new state s_{t+1};
- Update the table entry for $Q(s_t, a_t)$ as follows from (6) and (7);
- $s_t \leftarrow s_{t+1}$

Fig. 2. Pseudo – code of Q – learning algorithm

III. MATLAB IMPLEMENTATION OF Q – LEARNING ALGORITHM

One particular modification to presented Q – learning algorithm is the inclusion of λ $(0 \leq \lambda \leq 1)$ factor which was borrowed from temporal difference learning [3]. Higher the value, the lasting of the reward given in certain state is longer.

Every new epoch of the Q – learning algorithm should start from a randomly chosen point. In this case, it was more viable to randomly permute elements of the Q – value matrix. It gives the same effect as moving the agent to the next point and reconstructing the Q – value matrix, but this approach is better in terms of algorithm complexity.

There are two ways to obtain the maze matrix. First is to read it from the BMP file (monochrome). Second is to use maze generator developed for this purpose. It is a MATLAB implementation of a randomized Prim's algorithm for maze generation (python implementation can be found everywhere on the web including Wikipedia [10]).

As mentioned before, maze is described as a matrix with 0s and 1s for walls and empty spaces, respectively.

This implementation works with matrix representation of knowledge, and the algorithm for transforming the maze matrix into knowledge matrix is also implemented in MATLAB.

This implementation works with matrix representation of knowledge. The code for transforming maze matrix into knowledge matrix (Q – values) is also written in MATLAB syntax, and implements formation and initialization of matrix of Q – values based on connected graph creation method.

MATLAB script should output two figures showing the original maze, and the maze with noted path leading out of the maze from the starting point.

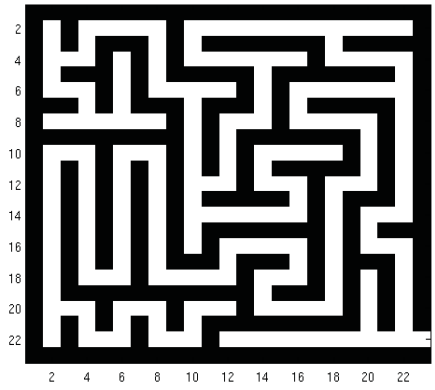

Fig. 3. Original maze

For the maze shown on Fig. 3 algorithm gives the second maze as shown on Fig. 4.

As one can see, the trajectory is marked with red coloured line, which connects top – left corner with bottom – right which is the maze solution as stated in Section I.

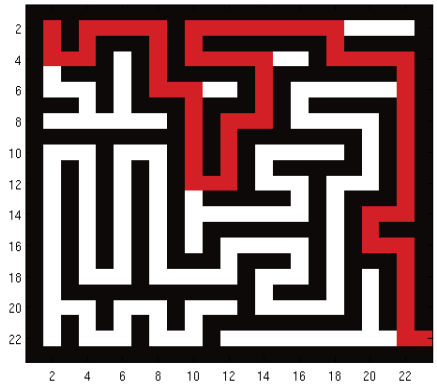

Fig. 4. The solution of the maze in Fig. 3. Red line marks the path leading to exit

IV. PERFORMANCE ANALYSIS

To analyse the performance of this algorithm specific test had to be devised.

Two variables were measured: time of execution, and total number of iterations. In theory, according to [11], general case of Q – learning algorithm should be of complexity $O(n)$. The other interesting aspect of the algorithm is the time of execution which is more important in real – time applications (i.e. autonomous robot navigation).

TABLE I. EMPIRICAL DATA FOR TIME OF EXECUTION TEST

Maze Size	Average Time of Execution (in seconds)
5x5	0.5426
7x7	0.8450
9x9	1.2712
11x11	2.5177
13x13	3.4441
15x15	6.3306
17x17	9.4527
19x19	17.1527
21x21	24.4530

TABLE II. EMPIRICAL DATA FOR NUMBER OF ITERATIONS TEST

Maze Size	Average Number of Iteration
5x5	2587.4
7x7	3972.5
9x9	5822.7
11x11	10755.4
13x13	13084.1
15x15	21139.9
17x17	26186.5
19x19	31978.1
21x21	38561.6

This experiment involved running the code for input mazes of different sizes (5x5, 7x7, 9x9, 11x11, 13x13, 15x15, 17x17, 19x19, and 21x21). For every maze, the algorithm was executer 10 times and the average was taken for both tests.

Results for both tests are given in Table I and II. Graphs for these tables are given in Fig. 5, and Fig. 6.

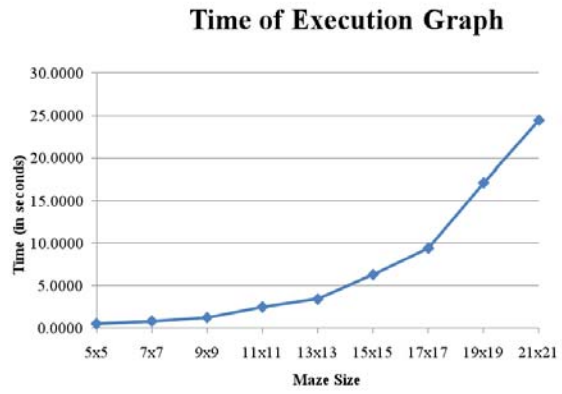

Fig. 5. Graph showing increase of time of execution with respect to maze size

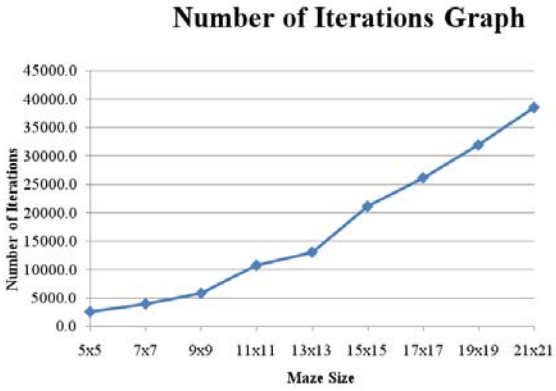

Fig. 6. Graph showing the increase in number of iterations with respect to maze size

As it is shown on Fig. 6, number of iterations linearly increases with the maze size which is consistent with the complexity analysis in [11]. On the other hand, the increase in time of execution is not linear, rather quadratic of cubic.

V. CONCLUSION

In this paper we presented the developed maze solving algorithm based on Q – learning algorithm. It was implemented using MATLAB, core and image processing toolbox, to be used with real robots in real time. The implementation is on abstract level so any problem which can be modelled as connected graph can be solved. The tests confirmed theoretical complexity analysis of the linearity of the algorithm but also showed that time of execution increases nonlinearly.

REFERENCES

[1] T. M. Mitchell, "Machine Learning," McGraw Hill, 1997.

[2] K.F. Lee, "Automatic Speech Recognition - The. Development of the SPHINX System" Kluwer Academic Publishers, Boston, 1989.

[3] G. Tesauro, "Temporal Difference Learning and TD-Gammon" Communications of the ACM 38, March 1995 / Vol. 38, No. 3

[4] J. de Lope, J. A. Martin, D. Maravall, "Applying Reinforcement Learning to Modular Cooperative Robotic Systems", Multidisciplinary Symposium on Reinforcement Learning (MSRL 2009) . Montreal, Canada. June 2009.

[5] M. McPartland, M. Gallagher, "IEEE Transactions on Computational Intelligence and AI in Games", vol. pp, no. 4, pp. 1, 2010.

[6] A. Albers, S. Schillo, D. Sonnleithner, M. Frietsch, P. Meckl, "A new two-layer reinforcement learning approach the control of a 2DOF manipulator", 2010 8th IEEE International Conference on Control and Automation (ICCA), Xiamen 2010, pp. 546 – 551.

[7] T. Nakashima, M. Udo, H. Ishibuchi, "Knowledge acquisition for a soccer agent by fuzzy reinforcement learning", IEEE International Conference on Systems, Man and Cybernetics, Washington DC, 2003, vol.5, pp. 4256 – 4261.

[8] C. J. Watkins, P. Dyan, "Q-learning. Machine Learning", vol.8, no. 3, pp. 279 – 292.

[9] C. J. Watkins, ""Learning from Delayed Rewards", PhD. Thesis, Cambridge, 1989.

[10] Wikipedia article on Maze Generation Algorithms, http://en.wikipedia.org/wiki/Maze_generation_algorithm.

[11] S. Koenig, R.G. Simmons, "Complexity Analysis of Real-Time Reinforcement Learning", In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 99-105, 1993.