Faster Finding of Optimal Path in Robotics Playground Using Q-Learning with “Exploitation-Exploration Trade-Off”

Faster Finding of Optimal Path in Robotics Playground Using Q-Learning with "Exploitation-Exploration Trade-Off"

Xiaosong Yuan*

Zhejiang University and University of Illinois at Urbana-Champaign United Institute, Zhejiang University, Haining, Zhejiang 314400, China

* Corresponding author's e-mail: Xiaosong.18@intl.zju.edu.cn

Abstract. It has been a challengeable task to use algorithms designed for specific situations to plan path in real-world environments, because of the uncertainty and unpredictability of the real world. However, learning algorithms have an effective adaptability to deal with real-world problems such as path planning. Because rules of how to work specifically are not preestablished but to be learnt. In this research, q-learning is modified with "Exploitation-Exploration Trade-Off" to find an optimal path faster in Robotics Playground, which is a real-world simulation in Simulink. "Exploitation-Exploration Trade-Off" is a strategy that enables q-learning to explore more parts of the environment so that it is able to learn more information within the same number of episodes. Four kinds of exploration probability are analyzed, which are linear (to episode), quadratic, hyperbolic and no exploration. The results show that the quadratic one has the best behavior among the four kinds of exploration probability when α and γ (two parameters in Bellman equation used to update q-table) are both 0.9. When α and γ are 0.9, the quadratic one is 27.41%, 32.53% and 36.66% faster than linear, hyperbolic and no exploration one with respect to the number of episodes used to find an optimal path. Effectiveness of balancing exploitation and exploration in q-learning is already proved in previous works, but the novelty of this research lies in testing specific exploration probabilities with different parameters (α and γ) and finding the most effective one as well as combining q-learning with real-world simulations. This research can help other q-learning-related works to save time and get a better result.

1. Introduction

It has been a challengeable task to plan a multi-environment-available path for mobile robots. If the working environments of robots are changed, engineers will have to revise the path planning codes or even rewrite. Workload will be large and efficiency will be low if environments are large or change frequently. However, in such cases, learning algorithms are able to work effectively with no need of changing codes.

Q-learning is a kind of reinforcement learning algorithm that can be used in real-world path planning. It seeks to find the best action to take under each state by gradually update q-table, which shows the reward of each action under each state and actions with higher reward will have priority to be taken [1]. Q-learning has a widely use in path planning nowadays, however, how q-learning can be used by robots with real wheels remains to be researched and there are potentials to improve q-learning to find an optimal path faster. Effectiveness of "Exploitation-Exploration Trade-Off" is researched in previous works, but which specific kind of exploration probability with which parameters should be used is not researched before this work. How to combine q-learning with real-world simulation also remains to be research pursuit.

This research aims to modify q-learning with "Exploitation-Exploration Trade-Off" so that the algorithm can find an optimal path faster. In the original q-learning, robot is only able to exploit action with highest reward under each state [2]. Environment information learnt by robot is limited so the learning process is not effective. "Exploitation-Exploration Trade-Off" is a strategy that permits robot to explore non-highest rewarded actions with a certain probability [3]. "Exploitation-Exploration Trade-Off" helps robot to explore more parts of the environment so that it is able to learn more information within the same number of episodes, which helps to improve the q-table and thus the final decisions.

This research focuses on solving two problems of q-learning. Firstly, the original q-learning chooses actions with highest rewards all the time, which can only exploit limited information of environments within each episode. This research will use "Exploitation-Exploration Trade-Off" to modify q-learning and find an effective exploration probability. Secondly, how q-learning can be used in real-world mobile robot simulation is still unclear. This research will extend q-learning to a real-world simulation to solve the second problem. This research focuses on real-world simulation rather than testing a physical robot because q-learning needs a mass of memory space and fast computing speed, which are beyond a microcomputer board's ability [4-5].

The average steps of the found path within certain number of episodes and how fast the algorithm converges to an optimal path are criteria of effectiveness. Q-learning with four kinds of exploration probability and different parameters will be tested. The four kinds of exploration probability are linear (exploration probability is linear to episode), quadratic (exploration probability is quadratic to episode), hyperbolic (exploration probability is quadratic to episode) and on exploration (the original q-learning). Parameters include α and γ in Bellman equation. Algorithm with less average steps and converging to an optimal path faster will be considered more effective.

The novelty of this research is implementing four specific kind of exploration probability and testing their effectiveness under different parameters (α and γ in Bellman equation). Therefore, this research is able to help later q-learning-related works to find a optimal result faster.

2. Related work

Path planning has been a significant and challengeable task in mobile robot design. Various path planning algorithms are designed, such as cell decomposition method, artificial potential method and so on [6]. However, such methods tend to be complicated and designed for certain maps, which lacks the ability to handle with the uncertainty in real-world map.

Reinforcement learning algorithms have been widely used in real-world path planning to deal with uncertainty and unpredictability of real world in the last few decades [7-9]. As one of the most widely used reinforcement learning algorithm, q-learning was firstly introduced by Chris Watkins in 1989 [2]. The convergence was proved by Watkins and Dayan in 1992 [10]. Both of which established the foundation of further research on q-learning, including this research. However, reinforcement learning takes a lot of time to train [11], including q-lerning. Previous works also did some research on exploitation-exploration balancing in q-learning. Richard Dearden and two coresearchers showed some properties of exploitation-exploration balanced q-learning, such as the q-value distributions and discussed several q-value updating rules [12]. Handy Wicaksono tested four learning rates (α = 0.25, 0.5, 0.75, 1) [13]. However, different kinds of exploration probabilities and subdivided combinations of α and γ are not researched so far.

Prof. Fumiya Iida from Bio-Inspired Robotics Lab (BIBR), Department of Engineering, University of Cambridge provided this research with codes that can find path in matrix mazes with q-learning without "Exploitation-Exploration Trade-Off" [3]. Codes that execute q-learning without "Exploitation-Exploration Trade-Off" in this research are modified from that part.

Simulation of real-world environment and demonstration of the moving of robots are based on Simulink in MATLAB, which is a graphical programming environment for the simulation of model-based design [14]. Robotics Playground is also used to help build environments for mobile robots, which is a collection of mobile-robot-related Simulink units made by MathWorks Student Competitions Team [15].

Previous work has established theory foundations and provided basic tools for this research. But how to combine theory with real-world simulation and make improvements to previous work remain to be the pursuit of this research.

3. Methods

Robots with wheels will be simulated in Robotics Playground to demonstrate the learnt path. As motor speed is limited, directly let robot run in simulation and learn a way by itself will take plenty of time. Generally, this research will let program generates a matrix corresponding to the environment in Robotics Playground and find a path in it firstly. Then, simulation will be started and the found path will be input to simulation. The robot will move along the input path. Details of how this section is arranged go as follows. Firstly, the original q-learning will be reviewed. Then, implementation of "Exploitation-Exploration Trade-Off" will be described. There will be three kinds of exploration probability, which are linear (to the current number of episode), quadratic (to the current number of episode) and hyperbolic (to the current number of episode). The effectiveness of those three kinds of exploration probability together with no exploration will be analyzed in the analysis section. Finally, the algorithm that makes mobile robot run in Robotics Playground will be described.

3.1. Q-learning

While planning path for mobile robots, design of an algorithm that can work in different environments is essential. Therefore, learning algorithms are widely used in mobile robot path planning. Among the many learning algorithms, q-learning is one of the most qualified. Q-learning is considered to be an off-policy algorithm because policies of what actions to take in different circumstances are not needed to be preestablished [1]. The original q-learning without "Exploitation-Exploration Trade-Off" will be introduced in this part. State, action, reward and q-table will be described to clarify q-learning. As the whole section works as a whole, some skills of making robots run in Robotics Playground such as establishing corresponding matrices will also be introduced here, so that the order of the program will not be reversed.

To achieve the goal of this research, the first thing to do is extracting environment information and establishing a corresponding matrix in MATLAB to speed up the learning process. Firstly, environment in Robotics Playground is divided into blocks with size 0.5 * 0.5. For example, the environment in the demonstration program has size 8 * 8, so it is divided into 16 * 16 = 256 blocks. The number of blocks on the length and width of the environment determines the rows and columns of the corresponding matrix. For example, the corresponding matrix in the demonstration program is 16 * 16 because there are 16 blocks on the length and width of the environment respectively. Entries in the matrix will be initialized to 0. Next, input the location and size of obstacles to program and initialize corresponding entries in matrix to -1. Finally, initialize the goal to 1. The establishment of corresponding matrix is done. Entries in the established matrix not only translates the environment in simulation to program, but also shows the reward of reaching each state, which will be clarified in the next paragraph.

With a corresponding matrix, the next thing to do is executing q-learning in it and learning a path. As q-learning is a kind of reinforcement learning, it also involves essential elements of reinforcement, which are an agent, a set of environments, a set of states and a set of actions [16]. In this research, the agent and environment are robot and map in Robotics Playground respectively. And as q-learning will be executed in matrix to speed up the learning process, the corresponding agent and environment are a coordinate recording the location of robot and the matrix. A state s is a coordinate in matrix. There are four actions a per state, which are left, down, right and up, labeled by 1 to 4 respectively. After an action a, the robot moves to the next state s'. For example, assume the current state is (2, 2). Then, if the robot takes action 1, it moves left and reach state (2, 1). Reaching each state s has a reward r. The reward r of reaching each state s is recorded in the established matrix. $r = -1$ means an obstacle is reached. $r = 1$ means the goal is reached. $r = 0$ means an available location.

The quality q of a state-action combination is stored in q-table [1]. Therefore, the q-table is a three-dimensional matrix Q. For example, if the reward of action 3 in state (6, 5) in a certain episode is 0.3,

it will be stored as $Q(6,5,3) = 0.3$. The algorithm will compare the reward of each action at the current state and choose action with highest quality q to take. The quality q of the chosen action will be updated by Bellman equation, as being shown in equation (1).

$$Q_{new}(s,a) = Q(s,a) + \alpha[r + \gamma \cdot max_{a'}Q(s',a') - Q(s,a)] \quad (1)$$

Then, the algorithm will move to the next state as being described in the last paragraph. When an obstacle or the goal is reached, i.e., when reward of taking highest-q action at current state is -1 or 1, the current episode is terminated and a new episode is started. An episode is the process of transiting from the initial state to a terminating state (obstacle or goal). The total number of episodes is preset before the loop of executing all episodes is started. After all episodes are done, taking actions with highest-q in q-table from the initial state will guide robot to the goal, if the number of episodes is enough.

Details of how q-learning works could be seen in the following flow chart (Figure 1).

3.2. "Exploitation-Exploration Trade-Off"

The original q-learning exploits actions with highest q all the time, which is only able to handle limited environment information. The learning speed is not effective enough. Therefore, this research uses "Exploitation-Exploration Trade-Off" to solve this problem. Exploitation means making use of known information of the environment to decide which action to take. The original q-learning only uses exploitation to learn the environment. Exploration means explore non-highest-q actions at a certain probability. Trade-Off means the program should make a balance between exploitation an exploration. With "Exploitation-Exploration Trade-Off", the algorithm will be able to randomly choose a non-highest-q action to take. There are three kinds of exploration probability in this research, which are linear (to the current number of episode), quadratic (to the current number of episode) and hyperbolic (to the current number of episode). The three kinds of exploration probability are shown in equation (2) (3) (4) respectively, where p represents exploration probability, t represents total number of episodes and e represents current number of episodes. Total number of episodes is preset and the only variable is current number of episodes t.

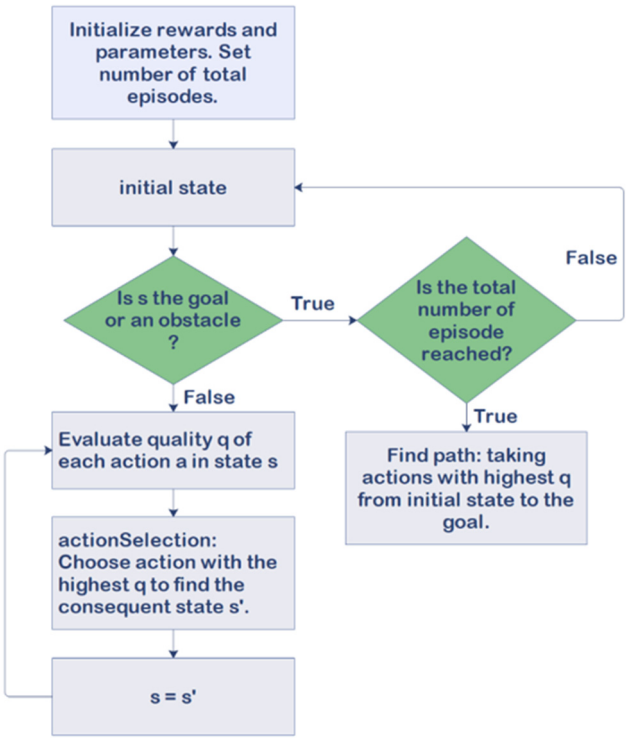

Figure 1. Flow chart of q-learning

$$p = -\frac{1}{t}e + 1 \quad (2)$$

$$p = \frac{1}{t^2}e^2 - \frac{2}{t} + 1 \quad (3)$$

$$p = \frac{1}{e} \quad (4)$$

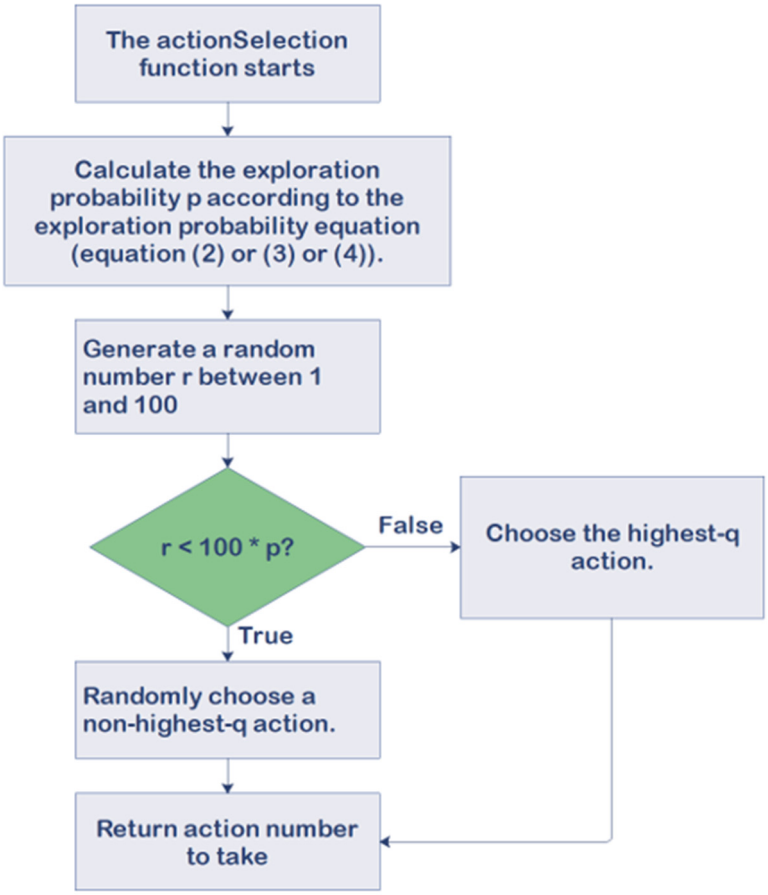

Figure 2. Flow chart of "Exploitation-Exploration Trade-Off"

Details of "Exploitation-Exploration Trade-Off" could be seen in the following flow chart (Figure 2). The flow chart shows how "Exploitation-Exploration Trade-Off" works in action selection. Replacing the $actionSelection$ part in Figure 1 with Figure 2 will enable q-learning to do "Exploitation-Exploration Trade-Off".

3.3. Move in Robotics Playground

To find a path in Robotics Playground, this research firstly extracts environment information from the map, as being described in part A of this section. Then, train the robot using q-learning or q-learning with "Exploitation-Exploration Trade-Off" to find a path in the corresponding matrix, as being described in part B and part C of this section. Finally, the robot moves in Robotics Playground along the found path, which will be described in this part. Robot in this research has two wheels and no sensors. It is able to detect the current location of itself in Robotics Playground.

After q-learning is done, the Simulink is started. The path found in the corresponding matrix is output to Simulink as a set of coordinates. There are four inputs to the control function of Simulink, which are the coordinate set of found path, the x position of the robot in the map $R1xPose$, y position of the robot in the map $R1yPose$ and the turned degrees θ of the robot $R1tPose$. There are two outputs from the control function, which are the motor speed of left and right motor.

Firstly, the found path is transformed back to the range of the small block in the map. Assume the robot starts from the left bottom corner of the map. Then, x corresponds to the row of the matrix and y corresponds to column. The corresponding range could be obtained by equation (5) and (6), where $pathFound$ denotes the found path from input.

$$-4 + 0.5(pathFound(i,1) - 1) \leq x \leq -4 + 0.5pathFound(i,1) \quad (5)$$

$$-4 + 0.5(pathFound(i,2) - 1) \leq y \leq -4 + 0.5pathFound(i,2) \quad (6)$$

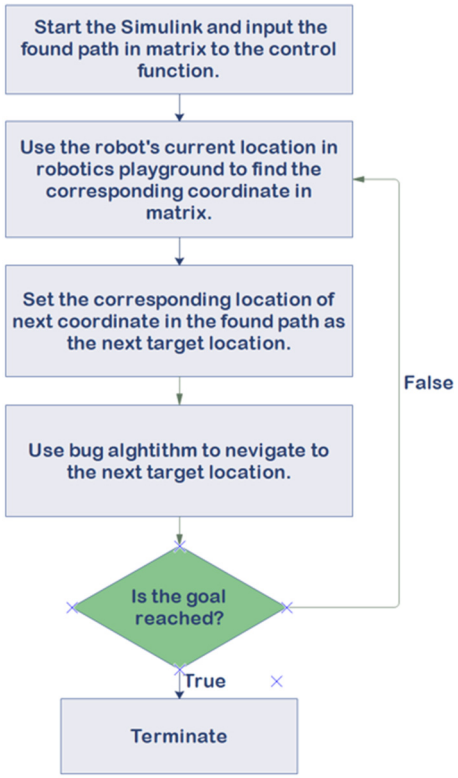

Figure 3. Flow chart of navigation program

Take the demonstration program as an example, the coordinate of the initial state is (1, 1). It means the robot starts from the left bottom block of the map (the corresponding matrix is rotated 90 degrees clockwise to make the robot start from (1, 1) rather than (16, 1)). Then, it should be within the range of $-4 \leq x \leq -3.5$ and $-4 \leq y \leq -3.5$. The robot compares its current (x, y) location with the range of blocks in the path and it will know where it is and what its next target location is. Methods of navigating from current block to the next block in the path will be introduced in the next paragraph.

The control function uses Bug algorithm to navigate between different blocks in the path [17]. The Bug algorithm is simplified because there is no obstacle within the found path. Therefore, the robot only needs to find out its current block (the last paragraph) and its next target block. The robot is able to find the center coordinate of the next block with equation (7), where $targetLocation$ denotes the coordinate of the center of next block in the path.

$$targetLocation = (\ 0.5pathFound(i+1,1) - 4.25,\ \ 0.5(pathFound(i+1,2) - 4.25)\ ) \quad (7)$$

Then, the robot uses simplified Bug algorithm to navigate from current location to the target location. The simplified Bug Algorithm works as following. If the turned degree of the robot is larger than the slope of the line connected by the current location and target location, turn right. If smaller, turn left. If equals to, then go straight. Details of how the navigation program (control function in Simulink) works could be seen in the following flow chart (Figure 3).

4. Results and analysis

In this section, four kinds of exploration probability together with different parameters in Bellman equation (equation (1)) are tested in order to find the most effective exploration probability and most effective parameters. The four kinds of exploration probability are shown in equation (2) (3) (4) in last section together with no exploration. Both α and γ in the Bellman equation are tested from 0.1 to 0.9 with an interval of 0.1. Each combination of exploration probability and parameters are tested 30 times.

Two criteria are used to determine the effectiveness of algorithms. The first criterion is average steps within certain number of episodes. In the testing code, the number of episodes is 5000. Algorithm that is able to find a path with fewer average steps is considered to be more effective. The second criteria is converge time, which means with how many episodes the algorithm can find an optimal path. The maximum number of episodes in 5000 in the test code and the optimal path has 31 steps. Algorithm with less converge time is considered to be more effective.

4.1. Effectiveness of algorithms with respect to α

The test results of the average steps within 5000 episodes and the converge time of algorithms with four kinds of exploration probability with respect to α are shown in Figure 4 and Figure 5.

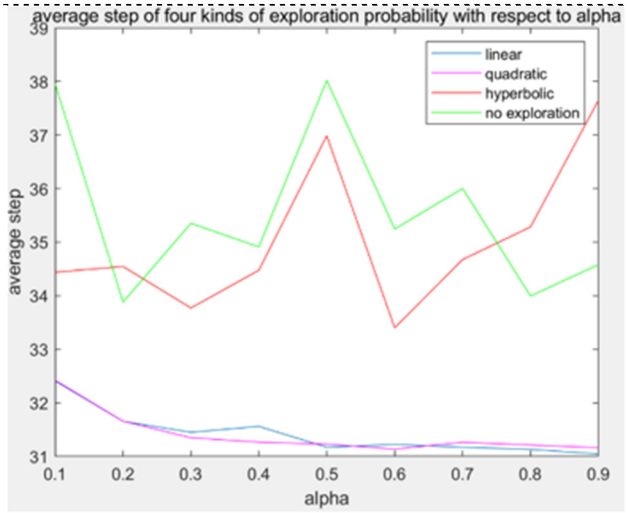

Figure 4. Average steps of algorithms with four kinds of exploration probability with respect to α.

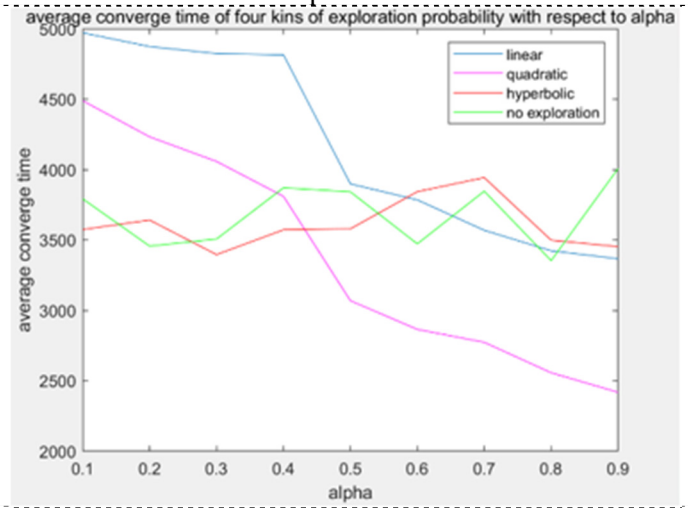

Figure 5. Converge time of algorithms with four kinds of exploration probability with respect to α.

Figure 4 shows that the quadratic one and linear one has fewer average steps than the hyperbolic one and no exploration one with respect to α. The average steps of the quadratic one and linear one are similar and decrease when α increases. There is no obvious rule for the hyperbolic one and no exploration one. Figure 5 shows that the converge time of the hyperbolic one and no exploration one is uniformly distributed with respect to α but the quadratic one and linear one decreases when α increases. Among four curves in Figure 5, the quadratic one has the least converge time when α is larger or equal to 0.5 and further decreases when α increases. Therefore, according to the two critera, the quadratic one is the most effective one among the four algorithms when α is around 0.9.

4.2. Effectiveness of algorithms with respect to γ

The test results of the average steps within 5000 episodes and the converge time of algorithms with four kinds of exploration probability with respect to γ are shown in Figure 6 and Figure 7.

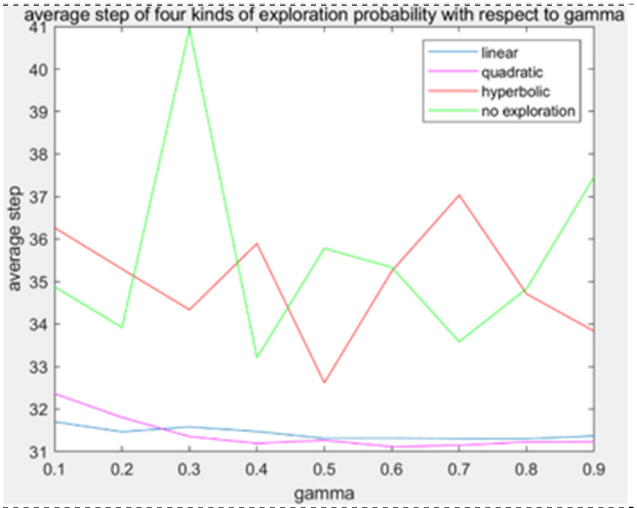

Figure 6. Average steps of algorithms with four kinds of exploration probability with respect to γ.

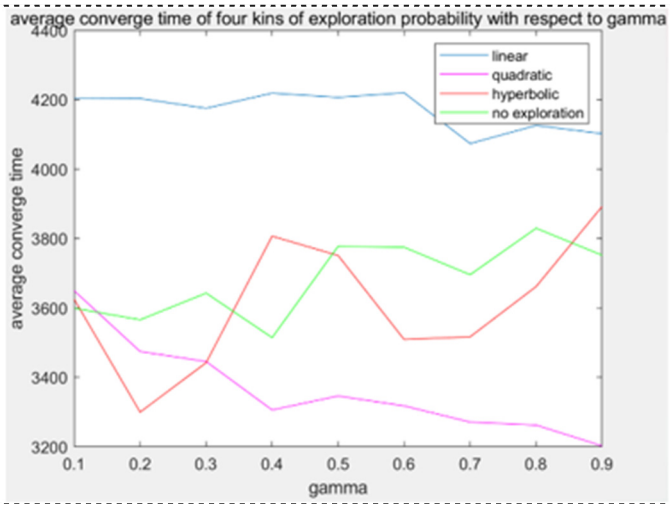

Figure 7. Converge time of algorithms with four kinds of exploration probability with respect to γ.

Figure 6 shows that the quadratic one and linear one has fewer average steps than the hyperbolic one and no exploration one with respect to γ. The quadratic one has an even fewer average steps than the linear one when γ is larger or equal to 0.3. There is no obvious rule for the hyperbolic one and no exploration one. The average steps of the quadratic one and linear one decrease when γ increases,

which means the quadratic one has the least average steps when α is around 0.9. Figure 7 shows that the converge time of the linear one is the largest and uniformly distributed with respect to γ. There is no obvious rule for the hyperbolic one and no exploration one. The converge time of the quadratic one becomes the least among four curves when γ is larger or equal to 0.3 and further decreases when γ increases. Therefore, according to the two critera, the quadratic one is the most effective one among the four algorithms when γ is around 0.9.

5. Conclusions

Previous section shows that q-learning with a quadratic exploration probability and with both α and γ equaling to 0.9 is the most effective one among the four kinds of algorithms. The average steps of the linear one, quadratic one, hyperbolic one and no exploration one is 31, 31.068966, 39.823529 and 37.666667 respectively when α and γ are both 0.9. The quadratic one is 21.98% and 17.52% less than the hyperbolic one and no exploration one. However, it is 0.22% larger than the linear one. This error occurs because one of the quadratic tests found a path with 2 more steps than the optimal one. This error is within a reasonable range that the two algorithms can be considered to have found equal average steps. The converge time of the linear one, quadratic one, hyperbolic one and no exploration one is 3316.5, 2407.4, 3568 and 3800.6 (unit: episodes) respectively when α and γ are both 0.9. The quadratic one is 27.41%, 32.53% and 36.66% faster than linear, hyperbolic and no exploration one. The quadratic one is the most effective one among the four kinds of algorithms and different parameters when α and γ are both 0.9.

This research contributes to other works by improving the effectiveness of q-learning. With a quadratic exploration probability, other q-learning-related works are able to learn an optimal result with less time by using the conclusions of this research.

There are also limitations in this research. Due to the limitation of research time, there are only 30 tests for each combination of exploration probability and parameters. More tests could be done in further works. More kinds of exploration probability with smaller parameter intervals could also be tested. This research is not able to explain the reason why the quadratic one if the most effective among the four kinds of exploration probability. Derivation of the most effective exploration probability could also be done in further works.

References

[1] Wikipedia, Q-learning [online]. https://en.wikipedia.org/wiki/Q-learning.
[2] Watkins C.J.C.H. (1989) Learning from Delayed Rewards, Cambridge University.
[3] Iida F. (2020) Foundations and Practice of Bio-Inspired Robotics in Lecture 4 [PDF], University of Cambridge, pp. 10-15.
[4] Hafner R, Riedmiller M. (2003) Reinforcement Learning on An Omnidirectional Mobile Robot, Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 1, Las Vegas, pp. 418 – 423.
[5] Anam K, Kuswadi S. (2008) Behavior Based Control and Fuzzy Q-Learning for Autonomous Mobile Robot Navigation, Proceeding of the 4th International Conference on Information & Communication Technology and Systems (ICTS).
[6] Gasparetto A, Boscariol P, Lanzutti A, Vidoni R. (2015) Path Planning and Trajectory Planning Algorithms: A General Overview, Motion and Operation Planning of Robotic Systems. Springer International Publishing, 3-27.
[7] Thombre P, (2018) Multi-objective Path Finding Using Reinforcement Learning, San Jose State University.
[8] Zhang J. L, Zhang J. Y, Ma Z, He Z. Z, (2017) Using Partial-policy Q-learning to Plan Path for Robot Navigation in Unknown Environment, Proc. 10th Int. Symposium on Computational Intelligence and Design (ISCID), vol. 1, pp. 85–90, Dec.

[9] Babu V, Krishna U, Shahensha S, (2016) An Autonomous Path Finding Robot Using Q-learning, Proc. Int. Conf. Intelligent Systems and Control, Coimbatore, India: IEEE, pp. 1–6.
[10] Watkins C.J.C.H, Dayan P, (1992) Q-learning, Machine Learning, 8, 279-292.
[11] Wu Z. F, (2018) Application of optimized q learning algorithm in reinforcement learning, Bulletin of Science & Technology, vol. 36, no. 2, pp. 74–76.
[12] Dearden R, Friedman N, Russell S, Bayesian Q-learning [online]. https://www.aaai.org/Papers/AAAI/1998/AAAI98-108.pdf.
[13] Wicaksono H. (2011) Q learning behavior on autonomous navigation of physical robot, 2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Incheon, pp. 50-54.
[14] Wikipedia, Simulink [online]. https://en.wikipedia.org/wiki/Simulink.
[15] MathWorks Student Competitions Team, Robotics Playground [online]. https://www.mathworks.com/matlabcentral/fileexchange/67157-robotics-playground.
[16] Wikipedia, Reinforcement Learning [online]. https://en.wikipedia.org/wiki/Reinforcement_learning.
[17] Iida F, (2020) Foundations and Practice of Bio-Inspired Robotics in Lecture 3 [PDF], University of Cambridge, pp. 27.