International Journal of Sciences: Basic and Applied Research (IJSBAR)

ISSN 2307-4531
(Print & Online)

http://gssrr.org/index.php?journal=JournalOfBasicAndApplied

The Amazing Race Repeated Update Q-Learning VS. Q-Learning

Hazem Bakkar[a*], Asma Q. Al-Hamad[b], Mohammad Bakar[c], Sohail Khan[d], Hassan Eleraky[e]

[a]*British University in Dubai, Dubai, UAE*

[b,c,d,e]*Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia*

[a]*Email: hazem.bakkar@hotmail.com,* [b]*Email: aqalhamad@iau.edu.sa,* [c]*Email: mwbakar@iau.edu.sa,* [d]*Email: sokhan@iau.edu.sa,* [e]*Email: haeleraky@iau.edu.sa*

Abstract

In this paper, we will conduct an experiment that aims to compare the performance of two reinforcement learning algorithms, the Repeated Update Q-learning algorithm (RUQL) [1] and the Q-learning algorithm(QL) [5]. A simulated version of a robot crawler developed by [6] will be used in this experiment, it is shown in figure (1). An investigation study about the difference in performance between RUQL and Q-learning algorithm (QL) [5] is discussed in this paper. Several trials and tests were conducted to estimate the difference in the crawler's movement using both algorithms. Additionally, a detailed description of the Markovian decision processes (MDPs) elements [2] is introduced, MDP model includes states, actions and rewards for the task in hand. The parameters that were used and tuned in this experiment will be mentioned and the reasons for choosing their values will be explained. Finally, the source code for the crawler robot was modified in order to implement RUQL and Q-Learning (QL) algorithms, Eclipse [3] and Java SE Development Kit 8 (JDK) [4] are used for this purpose. After running the crawler robot simulation, the results drawn from the experiment showed that RUQL significantly outperforms the traditional QL.

Keywords: Q-learning; Repeated Update Q-learning algorithm; Markovian decision processes; simulation.

---

* Corresponding author.

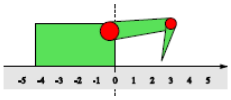
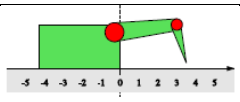
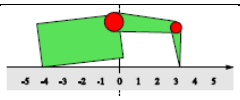
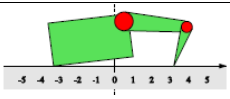
1. Introduction

The main task of this experiment is to give the crawler robot the ability to learn how to use its arm and hand in a way that will make the crawler move to the right as quick as possible using an improved Q-learning algorithm which is in this experiment RUQL. For achieving this purpose, two reinforcement-learning algorithms, namely, QL and RUQL were implemented in the simulation. Then the performance of both algorithms is analysed and results are concluded. This paper will include the following sections: MDP model for the task in hand, parameters used and tuned, techniques and improvements used in the experiment, and performance analysis for both QL and RUQL.

2. MDP model Definition

In this section, MDP model used in the experiment is detailed in table no. (1), while figure no. (1) is an approximate shape for the robot used in the simulation.

Table 1: MDP model adapted from [8].

Robot	Time	State		Reward	Action
		X	Y		
	0	Up	Left	0	Hand- right
	1	Up	Right	0	Arm- down
	2	Down	Right	0	Hand-left
	3	Down	Left	Number of pixels the robot moved > 0	Arm -up

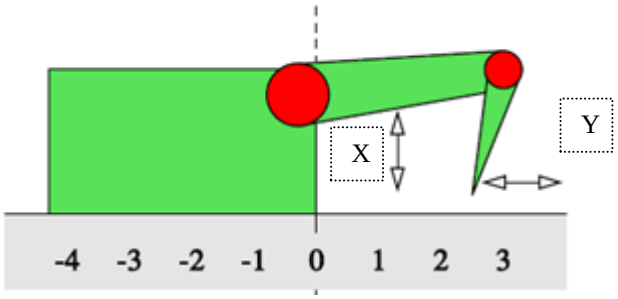


Figure 1: The robot used in the simulation, adapted from [8].

3. Parameter values used in the experiment

In this experiment, several tests were performed using variant values of the parameters that affect QL and RUQL performance [1]. This variation of parameters' values allowed us to see the differences in performance, these differences are clearly observed by manipulating each value of these parameters.

RUQL contains the same parameters that are used in the traditional Q-learning algorithm, these parameters are:

1- The learning rate parameter (α): This parameter gives weight for the new knowledge experienced by the robot, α value ranges between 0 and 1, where α = 1 means that only new knowledge will be considered, while α = 0 means that there is no learning at all, in other words new actions do not affect the learning process. In our experiment we used α with several values between 1 and 0.1, finally, we decided to keep α value equal to 1.

2- Discount rate parameter (γ): This parameter specifies the influence of the current reward on Q-matrix in the following steps. (γ) values range between 0 and 1. When γ= 0, the agent will consider the immediate reward ignoring the future steps, this will build a one-step learning sequence. However, when γ value goes closer to 1, this will build a learning sequence of many steps, which could delay the agent's learning process because the algorithm in this case will consider many future rewards acquired in many following steps. In our experiment, we used γ with value equal to 0.9.

3- Exploitation/exploration parameter (ε): This parameter gives the agent the ability to choose an action from the learned policy (exploitation), or experience a new action in learning step (exploration). In our experiment, we used ε-greedy action selection. We aimed to choose a random action related to the current state. Any action causes the robot to walk will be considered as a policy action, and its probability to be chosen again will be 1- ε, in the other hand, a new action will be chosen randomly with probability equal to ε. In our experiment, Epsilon is assigned a value equal to (0.3).

RUQL uses the following Equation [1]:

$$Q^{t+1}(s, a) = [1-\alpha]^{1/\pi(s,a)}Q^t(s, a) + [1-(1-\alpha)^{1/\pi(s,a)}][r + \gamma Q^t(s', a_{max})] \quad \textbf{Equation (1)}$$

There is another implementation for RUQL algorithm using the traditional QL algorithm [1]:

1. for $\frac{1}{\pi(s,a)}$ times do

2. Update Q(s, a) the traditional QL update equation:

3. $Q^{t+1}(s, a) = Q^t(s, a) + \alpha\,(r + \gamma\, max Q^t(s, a) - Q^t(s, a))$

4. end

This part of the pseudo code is guaranteed to represent RUQL correctly in the robot simulation. RUQL is also programmed using equation (1), but this representation needs more discussion and investigation in order to

correctly determine the value of $Q^t(s', a_{max})$ which exists in equation (1), results will determine whether equation no.(1) is correctly represented or not.

4. Techniques and improvements used in the experiment

In this experiment, we decided to use an enhanced version of Q-learning algorithm, which is RUQL algorithm [1]; RUQL overcomes the policy bias problem that appears in Q-learning algorithm especially when it is applied in the non-stationary environment. The policy bias problem is related to the probability of choosing actions; it describes the case when Q-learning estimates a low probability for choosing an action that is expected to give a high payoff. Q-learning has this problem because it focuses only on updating the selected actions that has a high probability of being selected [1]. The main Idea that makes RUQL overcomes the policy bias problem and outperforms QL is that RUQL assigns higher weights for the actions that have low probability to be selected, and assigns lower weights for the actions that have high probability of being chosen. This is a straight forward addressing of the policy bias problem that was described in the previous paragraph, RUQL is defining an opposite approach for the definition of the policy bias problem. RUQL update equation (equation (1)) is converting to the traditional Q-learning equation when choosing actions that have probability approaches to 1, in the other hand, RUQL will be equal to r + γQt(s', amax) when the probability of choosing a certain action approaches (0) [1]. Table no. (2) shows a part of the source code for QL algorithm in the original simulation [6] ,and two suggested and modified code for RUQL algorithm [1].

Table 2: Original and Modified code

Q-learning code in the simulation [6].
oldStateValue=Q[s1Old][s2Old][move]; aReward=reward(s1Old,s2Old,s1Cur,s2Cur); *Xpos+=aReward;* *Q[s1Old][s2Old][move]=oldStateValue+alpha*(aReward +gamma*getMaxQ(s1Cur,s2Cur) - oldStateValue);* *return aReward;*
RUQL code modification ***using loop***
oldStateValue=Q[s1Old][s2Old][move]; ////////////////no change is done here *aReward=reward(s1Old,s2Old,s1Cur,s2Cur); ////////////////no change is done here* *Xpos+=aReward; ////////////////no change is done here* ***for (int P= 0; P <= 1/getMaxQ(s1Cur,s2Cur); P++){*** ***Q[s1Old][s2Old][move]=oldStateValue+alpha*(aReward+gamma*getMaxQ(s1Cur,s2Cur)-oldStateValue);*** ***return Q[s1Old][s2Old][move]; }*** *return aReward; ////////////////no change is done here*
RUQL code modification ***using equation (1)***
oldStateValue=Q[s1Old][s2Old][move]; ////////////////no change is done here *aReward=reward(s1Old,s2Old,s1Cur,s2Cur); ////////////////no change is done here* *Xpos+=aReward; ////////////////no change is done here* ***double a1 = Math.pow(1- alpha, 1/getMaxQ(s1Cur,s2Cur));*** ***Q[s1Old][s2Old][move]= a1*oldStateValue+ (1-a1)*(aReward+gamma*getMaxQ(s1Cur,s2Cur));*** *return aReward; ////////////////no change is done here*

5. Software and packages used in this experiment

In this experiment, we used the following software and tools to perform our simulation:

1) Java SE Development Kit 8 (JDK), The JDK is a development environment for building applications, applets, and components using the Java programming language [4]. It can be downloaded using the following link: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.
2) Eclipse: Eclipse is an integrated development environment; it can customize the development environment using its powerful plug-in and workspace system that is included in its architecture [3]. Eclipse is written using Java and it can be used to develop applications. Eclipse can be used to develop applications in other programming languages like: Ada, ABAP,C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Natural, Perl, PHP, Prolog, Python, R, Ruby, Scala, Clojure, Groovy, Scheme, and Erlang [9] Eclipse can be downloaded from the following website http://www.eclipse.org/home/index.php.

6. Performance results using QL and RUQL

In this experiment, Both Algorithms were assigned the same values for the parameters α, γ and ε in testing the performance, the winning algorithm is determined by measuring the time taken by the robot to pass the track. The algorithm that records the least time to pass the track is considered the winner in the race. Table no. (3) shows part of several tests that were performed using the algorithms under study considering **γ = 0.9** in all the tests.

Table 3: Sample of the tests performed using QL, RUQL using loop, RUQL Equation (1)

ε	**α**	**Time-seconds/ QL**	**Time- seconds / RUQL/Loop**	**Time- seconds / RUQL/ Equation (1)**
0.1	0.1	102	103	128
0.2	0.1	121	120	97
0.3	0.1	154	144	141
0.4	0.1	154	154	333
0.5	0.1	188	256	225
0.7	0.1	317	317	375
0.1	1	78	99	98
0.2	1	89	89	92
0.3	1	140	78	107
0.4	1	132	94	115
0.5	1	158	120	131
0.7	1	201	176	192

Chart no. (1) and chart no. (2) are showing the results obtained after conducting the race several times using RUQL, QL and RUQL-equation (1).

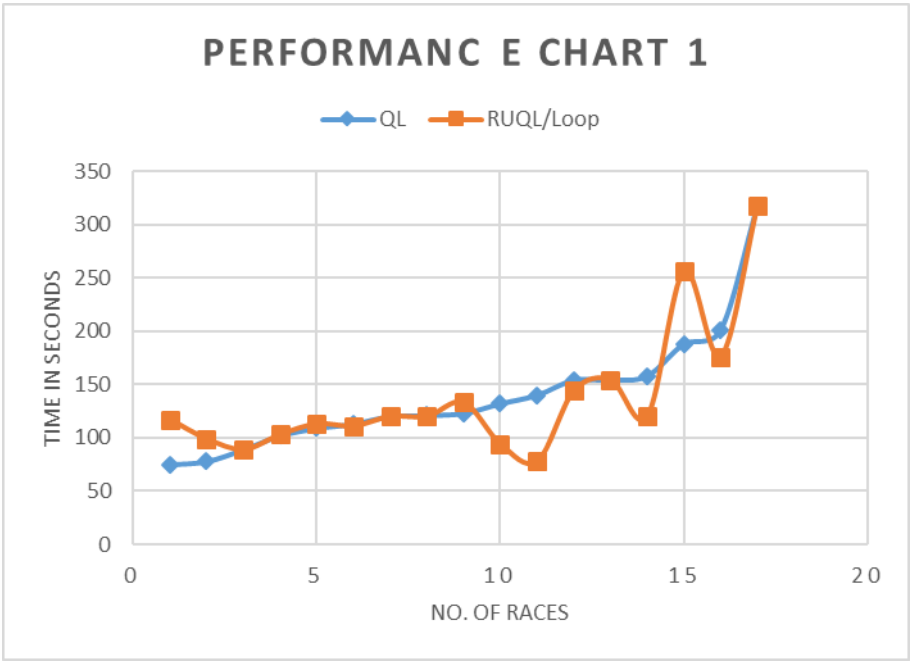


Figure 2

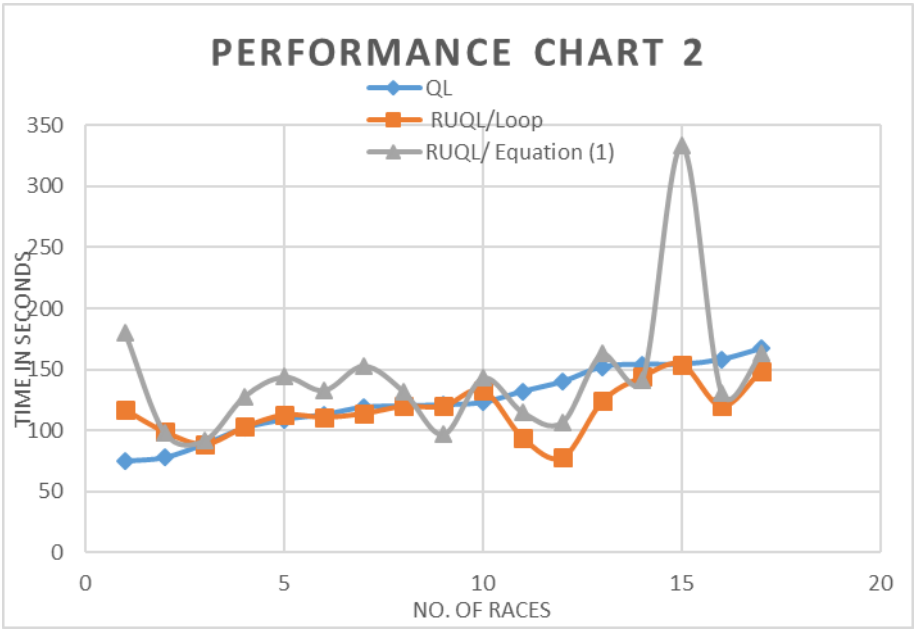


Figure 3

7. Conclusion

The charts show that RUQL outperforms QL and RUQL- equation (1), RUQL won 41% of the races, lost 35% of the races and achieved draw in 23% of the races. RUQL-equation (1) won only one race, it showed extremely slow performance, this leads to a conclusion that the update equation is not correctly implemented because of the vague value of a_{max}; therefore, it is not representing the original RUQL. The simulation best achieved record was 1 minute and 3 seconds, the race is uploaded on youtube.com [10], the following link leads to the race

uploaded file: http://www.youtube.com/watch?v=YOUYFaJZXm4.

8. Future work

In our seek to find the best values for the parameters in both QL and RUQL formulas, we suggested a fixed value for γ in the proposed experiment, which was 0.9, this is because we want to produce a multi-steps learning process, so that we can compare between QL and RUQL performance. In the future, we will experience assigning other parameters fixed values one by one and record the differences in performance between the algorithms under study. Also, we are planning to expand our experiment to include more improved versions of QL in the comparison like Hierarchical reinforcement learning algorithm [11].

References

[1]. Abdallah, S. and Kaisers, M. (2013). Addressing the Policy-bias of Q-learning by Repeating Updates. pp.1045--1052.

[2]. Bellman, R. (1957). A Markovian decision process.

[3]. Guindon, C. (2016). [online] Eclipse.org. Available at: http://eclipse.org [Accessed 01 June. 2016].

[4]. Oracle.com, (2016). Oracle | Hardware and Software, Engineered to Work Together. [online] Available at: http://oracle.com [Accessed 01 June 2016].

[5]. Watkins, C. and Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), pp.279--292.

[6]. Berghen, F. (2016). Kranf site: research. [online] Applied-mathematics.net. Available at: http://www.applied-mathematics.net [Accessed 01 June 2016].

[7]. Applied-mathematics.net, (2016). [online] Available at: http://www.applied-mathematics.net/qlearning/BotQLearning.java [Accessed 01 June 2016].

[8]. Tokic, M., Ertel, W. and Fessler, J. (2009). The Crawler, A Class Room Demonstrator for Reinforcement Learning.

[9]. Wikipedia.com, (2016). Retrieved 9 June 2016, from http://en.wikipedia.org/wiki/Eclipse_(software)

[10]. Youtube.com, (2016). YouTube. [online] Available at: http://youtube.com [Accessed 01 June. 2016].

[11]. Botvinick, M. (2012). Hierarchical reinforcement learning and decision making. Current Opinion In Neurobiology, 22(6), 956-962. http://dx.doi.org/10.1016/j.conb.2012.05.008.