M-Learning: A Computationally Efficient Heuristic for Reinforcement Learning with Delayed Rewards

Marlon Sneider Mora Cortes *, Cesar Andrey Perdomo Chary, Oscar J. Perdomo *

Posted Date: 29 July 2024

doi: 10.20944/preprints202407.2253.v1

Keywords: artificial intelligence; reinforcemente learning; heuristic; Markov Decision

 doi:10.20944/preprints202407.2253.v1

Article

M-Learning: A Computationally Efficient Heuristic for Reinforcement Learning with Delayed Rewards

Marlon Sneider Mora Cortes [1], Cesar Andrey Perdomo Charry [1] and Oscar J. Perdomo [3,*]

[1] Universidad Distrital Francisco José de Caldas, 111611, Colombia; msmorac@udistrital.edu.co (M.S.M.C.); cperdomo@udistrital.edu.co (C.A.P.C.)
[2] Universidad Nacional de Colombia, 111321, Colombia; ojperdomoc@unal.edu.co
* Correspondence: ojperdomoc@unal.edu.co

Abstract: The current design of reinforcement learning methods demands exhaustive computing. Algorithms such as Deep Q-Network achieved outstanding results in the development of the area. However, the need for thousands of parameters and training episodes is still a problem. Thus, this document proposes a comparative analysis of the Q-Learning algorithm (the inception to create Deep Q Learning) and our proposed method termed M-Learning. The comparison among algorithms using Markov decision processes with delayed reward as a general testbench framework. Firstly, a full description of the main problems related to implementing Q-Learning, mainly about its multiple parameters. Then, the foundations of our proposed heuristic with its formulation and the whole algorithm were reported in detail. Finally, the methodology chosen to compare both algorithms was to train the algorithms in the Frozen Lake environment. The experimental results and an analysis of the best solutions found that our proposed algorithm highlights the differences in the number of episodes necessary and their standard variations. The code will be available on a GitHub repository once the paper is published.

Keywords: reinforcement learning; agents; Q-Learning; Frozen Lake; heuristic

1. Introduction

The current development of artificial intelligence requires exhaustive computation [1], and the reinforcement learning (RL) area is no exception to this issue. Algorithms such as Deep Q-Network (DQN) [2] have achieved significant results in advancing RL by integrating neural networks. However, the requirement for thousands or millions of training episodes remains constant [3]. Training agents in parallel has proven to be an effective strategy [4] to reduce the number of episodes [5], which is feasible in simulated environments but poses an implementation challenge in real-world physical training environments.

In the context of reinforcement learning, Markov Decision Processes (MDPs) are typically the mathematical model used to describe the working environment [7]. MDPs have enabled the development of classical algorithms such as Value Iteration [8], Monte Carlo, SARSA, and Q-Learning, among others.

One of the algorithms most widely used is Q-Learning proposed by Christopher Watkins in 1989 [9]. Different approaches have been proposed based on the Q-Learning algorithm [10]. Its main advantage focuses on an optimal policy that maximizes the reward obtained by the agent if the action-state set is visited by a number infinitely many times [9]. The algorithm Q-Learning implemented in this work was taken from [9] and shown in Figure 1.

 doi:10.20944/preprints202407.2253.v1

Q-Learning

Algorithm parameters: step size $\boldsymbol{\alpha} \in (0, 1]$, small $\epsilon > 0$
Initialize Q(**s**,**a**), for all $\mathbf{s} \in \mathcal{S}^+$, $\mathbf{a} \in \mathcal{A}(\mathbf{s})$, arbitrarily except that $Q(\mathbf{s}_{terminal}, \cdot) = 0$
Loop for each episode:
- Initialize **s**
- Loop for each step of the episode:
- Choose **a** from **s** using policy derived from $Q(e.g.\, \epsilon\text{-greedy})$
- Take action **a**, observe $\mathbf{r}_t$, $\mathbf{s}_t$
-

$$Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \leftarrow Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) + \alpha \left[\mathbf{r}_t + \gamma \max_a Q(\mathbf{s}_t, \cdot) - Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})\right]$$

- until $\mathbf{s}_t$ is terminal

Some disadvantages of Q-Learning include the rapid table size growth as the number of states and actions [11] the problem of dimensional explosion. Furthermore, this algorithm tends to require a significant number of iterations to find a solution to a certain problem, and the adjustment of its multiple parameters can prolong this process.

Regarding the parameters of Q-Learning, the algorithm directly requires the assignment of the learning rate $\boldsymbol{\alpha}$ and the discount factor γ, which are generally tuned empirically through trial and error by observing the rewards obtained in each training session [12]. The learning rate $\boldsymbol{\alpha}$ directly influences the magnitude of updates made by the agent to its memory table. In [13], 8 different methods for tuning this parameter are explored, including iteration-dependent variations and changes based on positive or negative table fluctuation. Experimental results from [13] show that for the Frozen Lake environment, rewards converge above 10,000 iterations across all conducted experiments.

In Q-Learning, the exploration-exploitation dilemma is commonly addressed using the $\boldsymbol{\epsilon}$-Greedy policy, where one agent decides to choose between a random action and, the action with the highest Q-value with a certain probability. Current research around the algorithm focuses extensively on the exploration factor $\boldsymbol{\epsilon}$, as described in [14] and highlighted due to its potential to cause exploration imbalance and slow convergence [15–17].

The reward function significantly impacts the agent's performance as reported in various applications. For instance, in path planning tasks for mobile robots [18], routing protocols in unmanned robotic networks [19], and scheduling of gantry workstation cells [20], the reward function plays a crucial role by influencing the observed variations in agent performance. Regarding the initial values of the Q-table, the algorithm does not provide recommendations for their assignment. Some works as [21], which utilize the whale optimization algorithm proposed in [22], aim to optimize the initial values of the Q-table to address the slow convergence caused by Q-Learning initialization issues.

The reward function directly impacts the agent's training time. It can guide the agent through each state or be sparse across the state space, requiring the agent to learn from delayed rewards. As concluded in [23] when analyzing the effect of dense and sparse rewards in Q-Learning: "An important lesson here is that sparse rewards are indeed difficult to learn from, but not all dense rewards are equally good."

The heuristic proposed takes some conceptual ideas from Q-Learning along with its different proposals to search for alternatives focused on applications in real-physical environments. This heuristic allows for reducing the complexity of adjustment and time involved in the use of Q-learning. Moreover, the proposed algorithm, similar to Q-Learning, works on storage in a table and the relationship between states and actions through a numerical value. This algorithm is termed M-learning, and it will be explained in detail and referred to throughout the document.

 doi:10.20944/preprints202407.2253.v1

2. Definition of the M-Learning Algorithm

The main idea for the M-learning approach is to reflect on how an agent is considered to solve a task and what this implies in terms of the elements that produce the Markov processes. In contrast to Q-learning, this is not defined directly, rather it is based on the premise that the proposed reward function is consistent to be achieved and that consequently, the agent will seek to maximize the reward by reaching to complete the task eventually. This perspective merges two different problems into the same element to tune: the reward function and the way to get the solution.

In general, in the formulation of a task or problem and presented, the solution is implicitly defined or vice versa, if a solution is described, the problem is implicit. Although there is no notion of how to get there, like two sides of a coin in which you cannot always notice both sides. On the other hand, if you are searching to control the angle in a dynamic system, the solution will be the position state to be θ_{ref}. Now, when the approach of a task for an agent is thought within the framework of the MDP, the solution must also be described by its elements. Specifically, the solution to a task described as an MDP is a particular s of S, which we will call the objective state s_g.

In the reinforcement learning cycle, the reward signal aims to guide the agent in searching for a solution. Through it, the agent obtains feedback from the environment as to whether or not her actions are favorable. Q-learning uses a reward factor to modify the action-state and estimate its value. However, in environments where the reward is distributed dispersely and in which the agent most of the time does not receive feedback from the Q-learning environment, it must solve the search for the reward with time in exploration, reaching the exploitation-exploration dilemma, usually attacked with a policy of type ϵ-greedy.

The use of the ϵ-greedy policy, as a decision-making mechanism, requires the setting of the parameter of the decay rate ϵ_d, as the time that the agent will be able to explore looking for feedback from the environment, with a focus on reducing the number of iterations necessary to find a solution. This turns out to be decisive, as it is subject to the configuration before the training process and is not typical of the learning process, it is only possible to assign a favorable value by interacting with the problem through computation, making variations in the decay rate and number of iterations; keeping in mind that it is also conditioned by the rest of the parameters to be defined.

If the reward is thought of as the only source of stimulus for the agent in an environment with dispersed reward, the agent must explore the state space seeking to interact with states where it obtains a reward and, based on these, modify its behavior. As these states are visited more, the agent should be able to reduce the level of exploration and exploit more of its knowledge about the rewarded states. The agent explores in search of stimuli and to the extent that it acquires knowledge it also reduces arbitrary decision making. Exploration-exploitation can be understood discretely about each state in the state space. In this way, the agent can propose regions that it knows in the state space and, depending on how favorable or unfavorable, reinforce behaviors that lead it to achieve its objective or seek to explore other regions.

As stated, exploration and decision-making are strongly linked to the way the reward function is interpreted. The specific proposal in M-Learning consists of first limiting the reward function to values between -1 and 1, assigning the maximum value to the reward for reaching the goal state s_g. Second, each action in a state has a percentage value that represents the level of certainty about the favorableness of that action and, therefore, the sum of all actions in a state is 1. The M-Learning algorithm is shown below.

 doi:10.20944/preprints202407.2253.v1

M-Learning

Initialize Q(**s**,**a**), for all $\mathbf{s} \in \mathcal{S}^+$, $\mathbf{a} \in \mathcal{A}(\mathbf{s})$, with $\frac{1}{\mathbf{n_a|s}}$, and set $Q_{lim} < 1$
Loop for each episode:
- Initialize **s**
- Loop for each step of the episode:
- Choose **a** from **s** using probability Q(**s**,$\mathbf{a}_i$) for $\mathbf{a}_i$
- Take action **a**, observe $\mathbf{r}_t$, $\mathbf{s}_t$
- Update $Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$

$$Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \leftarrow Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})(1 + \mathbf{k})$$

- Resize probability of $\mathbf{s}_{t-1}$ to 1,

$$Q(\mathbf{s}_{t-1}, \mathbf{a}_i) \leftarrow \frac{Q(\mathbf{s}_{t-1}, \mathbf{a}_i)}{\sum_{j=1}^{n_a} Q(\mathbf{s}_{t-1}, \mathbf{a}_j)}$$

- until $\mathbf{s}_t$ is terminal

The first step in the M-Learning algorithm is the initialization of the table, unlike Q-learning, where the initial value can be set arbitrarily. The value of each state-action is initialized as a percentage value of the state according to the number of shares available. In this way, the initial value for each action of a state will be the same as shown in (1).

$$\begin{aligned} \sum_{i=1}^{n_a} Q(\mathbf{s}_j, \mathbf{a}_i) &= 1; \\ Q(s_j, a_i) &= \frac{1}{n_a}; \quad i = 1, 2, 3, .., n_a \end{aligned} \tag{1}$$

Once the table is initialized, the cycle for the episodes begins; the initial state of the agreement agent is then established with observation of the environment. The cycle of steps begins within the episode and, based on the initial state of the agent, it chooses an action to take. To select an action, the agent evaluates the current state according to its table, obtains the values for each action, and it selects one at random using its value as its probability of choice. Thus, the value of each action represents its probability of choice for a given state, as presented in equation (2).

$$P(\mathbf{a}_i|\mathbf{s}_j) = Q(\mathbf{s}_j, \mathbf{a}_i) \tag{2}$$

After selecting an action, the agent acts on the environment and observes its response in the form of the reward and the new state. When the agent has already passed to the new state, evaluate the action taken and modifies the value of that action-state according to (3).

$$Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \leftarrow Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})(1 + \mathbf{k}) \tag{3}$$

The percentage value of modification **k** of the value of the previous state-action s_{t-1} is computed under the existence consideration of the reward (4), if $\boldsymbol{r}_t$, exists, **k** is calculated based on the current reward or the current and previous reward. If there is no reward stimulus, it is calculated as the differential of information contained in the current state and previous ΔI.

$$K = \begin{cases} f_r & si \quad |r_t| \neq 0 \\ \Delta I & si \quad r_t = 0 \end{cases} \tag{4}$$

 doi:10.20944/preprints202407.2253.v1

When the environment returns a reward to the agent for his action, $|r_t| \neq 0$, r_t is a measure of the approach or distance concerning s_g, , in specific, the best case of the new state will be the target state $s_t = s_g$ with maximum positive reward $r_t = 1$, in the other case, the new state will be a state radically opposite to the objective state and its reward will be maximum but negative $r_t = -1$. For cases where the reward is not maximal, the value of f_r may be calculated by a function that maps the reward differential within the range of $[-1, 1]$, for this case, a first-order relation is proposed.

$$f_r = \begin{cases} r_t & si \quad |r_t| = 1 \\ \frac{r_t - r_{t-1}}{2} & si \quad 0 < |r_t| < 1 \end{cases} \tag{5}$$

If the reward function of the environment is described inform such that its possible values are 1, 0, and -1; the calculation of k can be rewritten by replacing f_r with r_t. This representsenvironments where it is difficult to establish a continuous signal that correctly guides the agent through the state space to reach your goal. In these situations, it may be convenient to only generate a stimulus on the agent by achieving the objective or cause the end of the episode. According to this, the Equation (4) can be rewritten as (6).

$$K = \begin{cases} r_t & si \quad |r_t| \neq 0 \\ \Delta I & si \quad r_t = 0 \end{cases} \tag{6}$$

When the environment does not return a reward $r_t = 0$, the agent evaluates the correctness of its action based on the difference in information stored for its current and previous state, as depicted in Equation (11). This is the way the agent spreads the reward stimulus.

$$\Delta I = I(s_t) - I(s_{t-1}) \tag{7}$$

To calculate the information of a state s_j, the difference between the value of the maximum action and the average of the possible actions in that state is evaluated in Eq. (8). The average value of the stock is also its initial value, which represents that there is no knowledge in that state that the agent can use to select a stock. In other words, the variation concerning the mean is a percentage indicator of the level of certainty that the agent has in selecting a certain action.

$$I(s_j) = max(s_j) - \frac{1}{n_a|s_j} \tag{8}$$

Since the number of shares available for the previous and current state may be different, which modifies the average value of the shares, it is necessary to put them on the same scale to be able to compare them. When the average value of the shares is established, its maximum variation of information is also established, this is how much is needed in the value of an action to have a probability of 1. The maximum information is then the difference between the maximum value and the average share value (9).

$$Imax_{s_j} = 1 - \frac{1}{n_a|s_t} \tag{9}$$

Since we want to modify the previous state-action, we adjust the information value of the current state to the information scale of the previous state, using (10).

$$I'(s_t) = I(s_t) \frac{Imax_{s_{t-1}}}{Imax_{s_t}} \tag{10}$$

It is possible that for a state s_t all its actions have been restricted in such a way that the value of all its shares possible is zero making it impossible to calculate the maximum information of s_t. For this

 doi:10.20944/preprints202407.2253.v1

reason, a negative constant value is established to indicate that the action taken leads to a state that we want to avoid. With this, the information differential is defined according to (11).

$$\Delta I = \begin{cases} I'(s_t) - I(s_{t-1}) & si \quad \sum_{i=1}^{n_a} Q(\mathbf{s}_t, \mathbf{a}_i) = 1 \\ -Q_{lim} & si \quad \sum_{i=1}^{n_a} Q(\mathbf{s}_t, \mathbf{a}_i) = 0 \end{cases} \tag{11}$$

The magnitude Q_{lim} is a parameter that controls the level of maximum saturation of an action due to propagation. With this value, it is possible to control how close the value of an action can be to the possible extremes 0 and 1.

After calculating k and making the modification in the action-state $Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \leftarrow Q(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})(1 + \mathbf{k})$ must be distributed the probabilities within the state to maintain $\sum_{i=1}^{n_a} Q(\mathbf{s}_{t-1}, \mathbf{a}_i) = 1$ applying (12) for state actions previous.

$$a'_i = \frac{a_i}{\sum_{i=1}^{n_a} Q(\mathbf{s}_{t-1}, \mathbf{a}_i)} \quad i = 1, 2, 3, .., a_n | s_{t-1} \tag{12}$$

In this way the variation in the value of the selected action over the rest of the actions. The beginning What is established is that the agent, having greater certainty of an action, reduces the possibility of selection of the other actions, or, on the contrary, by reducing the possibility of selecting an action, increases the level of certainty about the rest of the actions.

Similar to the ϵ-greedy, if you want to maintain a level minimum exploration value, it is necessary to restrict the maximum value that an action can achieve. This is established by the parameter Q_{lim}„ by conditioning the calculation of the redistribution of probabilities for the case in which the maximum action exceeds the saturation limit; ; $max(s_{t-1}) > Q_{lim}$, if this happens, it reduces the value of $max(s_{t-1})$ in such a way that when applying (12) it becomes the saturation limit Q_{lim}.

$$max'(s_{t-1}) = \frac{Q_{lim}}{1 - Q_{lim}} \sum_{i=1}^{n_a} Q(\mathbf{s}_{t-1}, \mathbf{a}_i) - max(s_{t-1}) \tag{13}$$

Finally, we return to the beginning of the cycle for the number of steps and continue executing the algorithm until reaching the limit of steps or reaching a final state.

3. Experimental Setup

In order to compare the performance of M-Learningagent training is carried out in front of Q-Learning using both algorithms in two different environments. Both groups of agents will have the same number of episodes of training and the same step limit per episode.

To measure the performance of the agents, a test is carried out Finish each episode until you complete the episode limit. During the episode, the agent will be able to modify his table according to her algorithm. At the end of the episode, either by reaching an end state or reaching the limit of steps per episode, The agent goes to the test stage where it cannot modify the values from her table. In the test stage, the agent plays 100 episodes with the table obtained at the end of the episode training. The policy applied in both agents for theselecting an action in the test is always taking the action with a higher $max(s)$ value. In this way, we seek to measure the agent's learning process and its ability to maintain the knowledge acquired.

The selected environment is Frozen Lake, Figure 1, which It consists of crossing a frozen lake without falling through a hole. Frozen Lake is available at the Gymnasium bookstore, which is a maintained fork of OpenAI's Gym [24] library which they define as "An API standard for learning for reinforcement." This environment has two variations which represent two different problems for the agent, In the first case, the environment allows the agent to move in the desired direction, and the state transition function of the environment is deterministic; In the second case when the agent wants

 doi:10.20944/preprints202407.2253.v1

to move in a certain direction environment may or may not lead you in that direction with a certain probability, the state transition function of the environment It is stochastic.

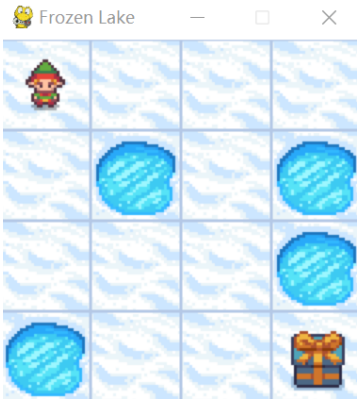

Figure 1. Initial Frozen Lake environment

In the Frozen Lake environment, the agent receives a reward of 1 for reaching the other end of the lake, -1 for falling into a hole, and 0 for the rest of the states. For the stochastic environment, the agent will move in the desired direction with a probability of $1/3$; otherwise, it will move in any direction perpendicular with the same probability of $1/3$ in both directions. Table 1 summarizes the configuration of the experimental methodology.

Table 1. Hyperparemteres used in the experiments

item	value
Number of agents	200
Episode limit	500
Step Limits	500

3.1. Parameter Tuning

To train agents using Q-Learning it is necessary to set the parameters of the algorithm. The initial values of the table are set as a pseudo-random value from 0 to 1. The value of ϵ_{min} for the ϵ-greedy policy is set to 0.005. Since α, γ and ϵ_d directly impact the performance of the algorithm, a mesh search is performed for these 3 parameters.

- ϵ_d: 0.8, 0.987, 0.996, 0.998
- α: 0.1, 0.3, 0.5, 0.7, 0.9
- γ: 0.1, 0.3, 0.5, 0.7, 0.9

The training of agents using our proposed M-learning method requires the setting of $Q_{lim} = 0.96$ parameter.

4. Results

A set of data is obtained with the scores of each agent to the test carried out after every episode. For each particular test, there are 200 scores per episode, for each episode the value is averaged of the scores and also their 95% confidence interval, assuming a normal distribution. The results obtained are displayed as line graphs, plotting the average value in a strong color and the confidence interval as the area around the solid line with a stronger color faint.

4.1. Deterministic Environment

4.1.1. Q-Learning

Figure 2 shows the results for the agents trained with the Q-Learning algorithm and the grid-search of its paramters. In specific, the results presented by 20,000 agents in the Frozen Environment

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 July 2024 doi:10.20944/preprints202407.2253.v1

Deterministic Lake are reported. The main observation was the increase of the ϵ_d, which is equivalent to the increase in exploration time, it causes agents to require more episodes to find a policy capable of solving 100% of the tests. The smaller values of α increase the dispersion in the agent scores and the increase in γ reduces the time it takes for agents to achieve the goal, 100 times the test.

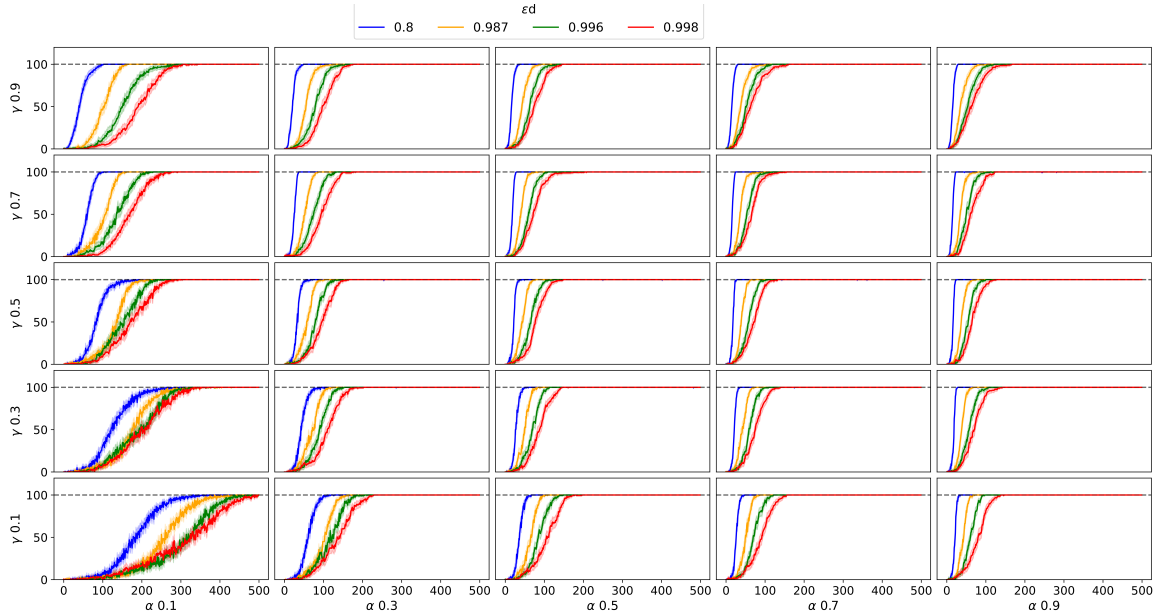

Figure 2. Q-Agents results deterministic environment.

The criterion for selecting a set of parameters as the best, it is proposed to take the combination with the least number of episodes required for its average value to Achieve 100% of the test tests. Based on this criterion, shown in Table 2 the results for $\epsilon_d = 0.8$ are ordered from largest to smallest.

Table 2. Set of parameters with the lowest number of episodes to achieve a score of 100 in the test.

Parameters			Test Episode
α	γ	ϵ_d	100%
0.1	0.1	0.8	331
0.1	0.3	0.8	262
0.1	0.5	0.8	179
0.3	0.1	0.8	110
0.1	0.7	0.8	98
0.1	0.9	0.8	98
0.3	0.3	0.8	86
0.5	0.1	0.8	67
0.3	0.5	0.8	56
0.5	0.3	0.8	52
0.3	0.9	0.8	49
0.7	0.1	0.8	46
0.3	0.7	0.8	37
0.5	0.5	0.8	37
0.5	0.9	0.8	35
0.7	0.3	0.8	32
0.7	0.9	0.8	31
0.9	0.1	0.8	31
0.5	0.7	0.8	29
0.9	0.9	0.8	29
0.7	0.5	0.8	28
0.9	0.3	0.8	28
0.7	0.7	0.8	25
0.9	0.5	0.8	25
0.9	0.7	0.8	24

 doi:10.20944/preprints202407.2253.v1

Figure 3 shows the behavior of the exploration inmesh for $\epsilon_d = 0.8$ according to the proposed criterion. It is noted how the best results are located in the value region high for parameters α and γ.

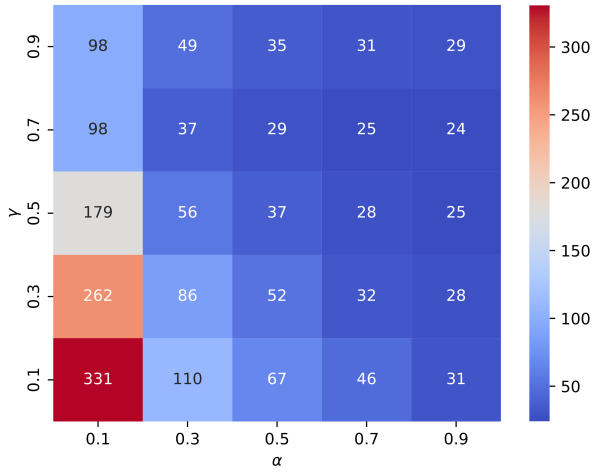

Figure 3. Set of parameters with the less number of episodes to scores 100 in the test.

The best set of parameters corresponds to α: 0.9, γ: 0.7, ϵ_d: 0.8, where the average of the agents trained with Q-Learning obtained a score of 100 for the test in episode 24. Figure 4 shows the training result of 1000 agents, using the best set of parameters, together with its standard deviation per episode. By increasing the number of agents, with the same set of parameters, Q-Learning obtained a score of 100 for the test in episode 26 for all agents.

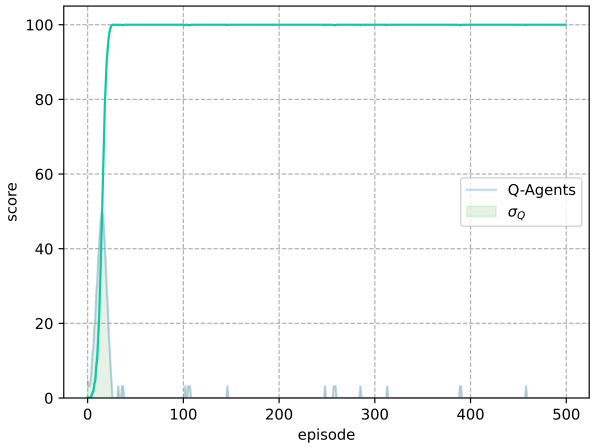

Figure 4. Set of parameters with the lowest number of episodes to achieve a score of 100 in the test.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 July 2024 doi:10.20944/preprints202407.2253.v1

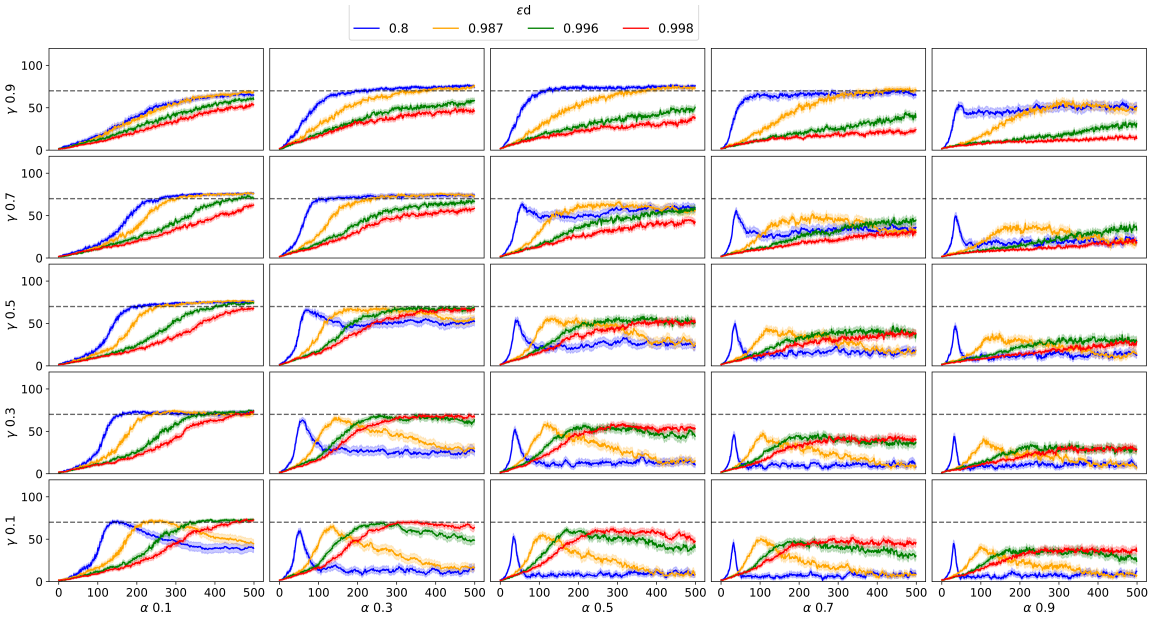

Figure 5. Q-Agents results stochastic environment.

4.1.2. M-Learning

Figure 6 shows the results obtained per 1000 agents trained with the M-Learning algorithm in the environment Frozen Lake deterministic. It can be seen how, during the 500 episodes, the average groups the behavior of the 1000 agents. The average score reaches 100% of the test in episode 18, the standard deviation is also included per episode of results.

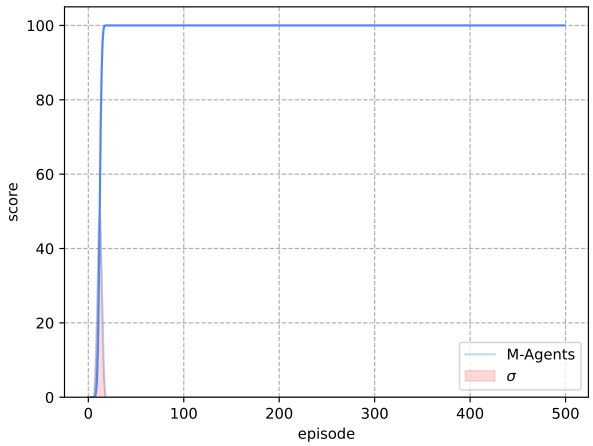

Figure 6. M-Agents results deterministic environment.

4.2. Stochastic Environment

4.2.1. Q-Learning

In Figure 5 shows the results obtained for the agents trained with Q-learning and the grid search of its parameters. In total, the results obtained by 20000 agents in the stochastic Frozen Lake environment are shown. It can be seen that the increase in ϵ_d increases the exploration time, this causes the agents to require more episodes to find a stable policy. Unlike deterministic Frozen Lake, it is not possible to achieve 100% of test tests due to the nature of the environment.

As a criterion for selecting a set of parameters, two conditions are proposed: the first, taking the combination with the lowest number of episodes required so that its average value exceeds 70% of the test tests, and the second, that during the rest of episodes, its average is not below 65. Based on this

 doi:10.20944/preprints202407.2253.v1

criterion, the results that meet the first condition are displayed in the Table 3 ordered from highest to lowest.

The best set of parameters corresponds to α: 0.3, γ: 0.7, ϵ_d: 0.8, where the average of the agents trained with Q-Learning obtained a higher score to 70 for the test in episode 101. Figure 7 shows the training result of 1000 agents, using the best set of parameters, along with their standard deviation per episode. By increasing the number of agents, with the same set of parameters, Q-Learning obtained a score above 70 for the test in episode 120.

Table 3. Parameter sets with the lowest number of episodes exceeding the score of 70 without being below 65 in the rest of the test episodes.

Parameters			Test Episode
α	γ	ϵ_d	100%
0.3	0.3	0.998	485
0.1	0.7	0.996	459
0.1	0.3	0.998	453
0.1	0.1	0.998	446
0.1	0.5	0.996	401
0.1	0.3	0.996	358
0.5	0.9	0.987	357
0.7	0.9	0.987	351
0.1	0.1	0.996	335
0.3	0.9	0.987	315
0.1	0.7	0.987	303
0.1	0.7	0.8	259
0.1	0.5	0.987	250
0.1	0.3	0.987	239
0.3	0.7	0.987	236
0.3	0.9	0.8	194
0.1	0.5	0.8	193
0.1	0.3	0.8	156
0.5	0.9	0.8	118
0.3	0.7	0.8	101

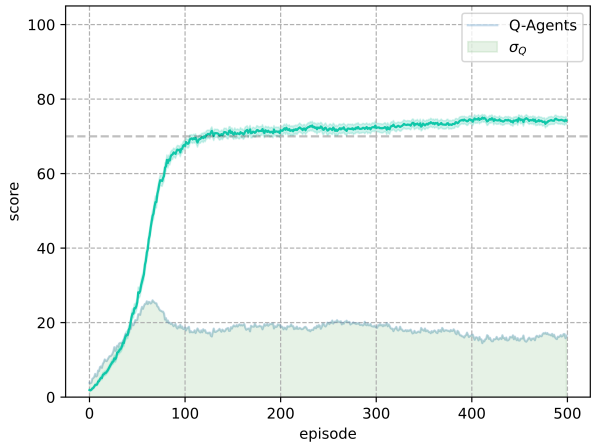

Figure 7. Set of Q-Learning parameters with the lowest number of episodes to achieve a score above 70 in the test.

4.2.2. M-Learning

In Figure 8 presented the results obtained by 1000 agents trained with the M-Learning algorithm in the stochastic Frozen Lake environment. It can be seen how during the 500 episodes the average groups the behavior of the 1000 agents, the average score exceeds the 70% of the test in episode 46, and the standard deviation per episode of the results is also included.

 doi:10.20944/preprints202407.2253.v1

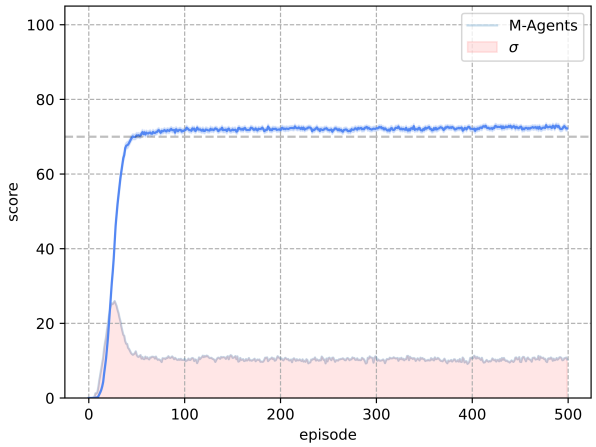

Figure 8. Obtained results with M-Agents stochastic environment.

5. Discussion

5.1. Deterministic Environment

In Figure 9 presents the best solutions obtained by the agents trained with Q-Learning and M-Learning, along with their standard deviations per episode. It can be seen how M-Learning first reaches the score of 100 with a smaller standard deviation area than Q-Learning. It can also be seen that Q-Learning, by continuing the learning process, may at some point have losses in the acquired knowledge and present a score less than 100, even when the environment remains the same.

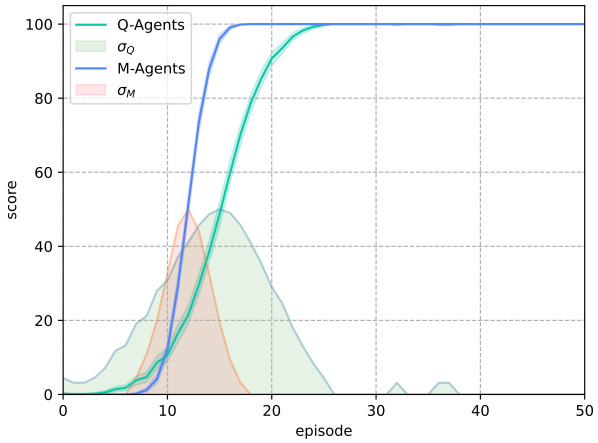

Figure 9. Deterministic environment results for the Q-Lerning and M-Lerning algorithms.

The frequency with which the agents achieve the objective for each episode may be analyzed, based on the evaluation criterion and the episode where the agents achieve it. This is shown in Figure 10, which illustrates the number of agents who achieved a score of 100 on the test in each episode.

The agents trained with Q-Learning achieved the goal on average in episode 13,827 with a standard deviation of 4.18 episodes, while the agents trained with M-Learning took an average of 12,456 episodes to reach the goal, with a standard deviation of 1.74 episodes. This represents a 9.91% reduction in the average number of episodes needed to achieve the goal and a 58.37% reduction in the standard deviation of episodes from the mean.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 July 2024 doi:10.20944/preprints202407.2253.v1

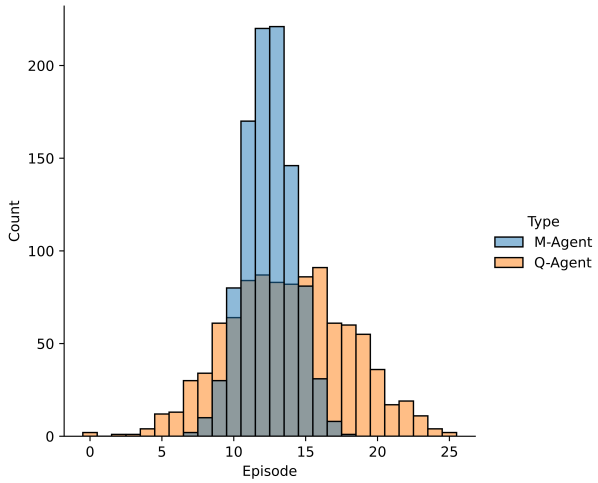

Figure 10. Deterministic environment results in distribution for the Q-Learning and M-Learning algorithms.

5.2. Stochastic Environment

In Figure 11 shows the best solutions obtained by the agents trained with Q-Learning and M-Learning, along with their standard deviations.

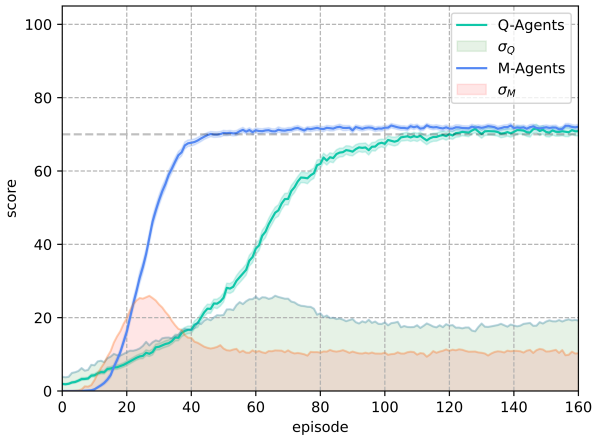

Figure 11. Stochastic environment results for the Q-Lerning and M-Lerning algorithms.

It can be seen how M-Learning first surpasses the score of 70 with a smaller standard deviation area than Q-Learning. It can also be seen that Q-Learning, by continuing the learning process, converges with a higher standard deviation than M-Learning.

In Figure 12 attention is focused on the evaluation criterion and the episode where the agents reach it. For each episode, the number of agents who achieved a score greater than 70 in the test is depicted.

The agents trained with Q-Learning achieved the objective on average in episode 57,591, with a standard deviation of 14.77 episodes, while the agents trained with M-Learning took 31,186 episodes on average to achieve the objective with a standard deviation of 7.42 episodes, reducing the average number of episodes to achieve the objective by 45.84% and the standard deviation of episodes concerning the mean by 49.75%.

 doi:10.20944/preprints202407.2253.v1

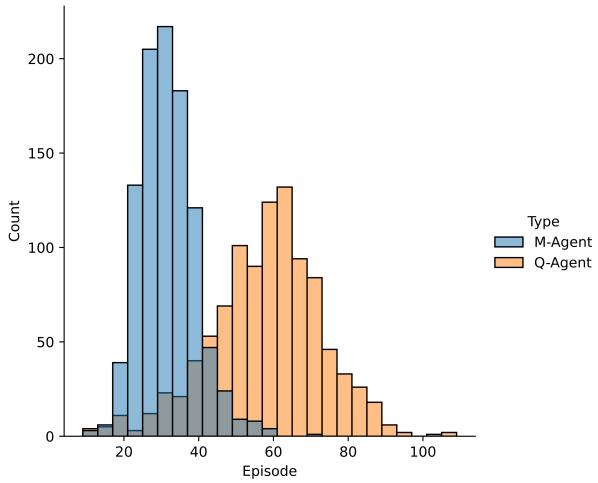

Figure 12. Stochastic environment results in distribution for the Q-Learning and M-Learning algorithms.

6. Conclusions

The use of reinforcement learning for an unknown problem using Q-Learning, requires carrying out a broad exploration of its parameters, as presented in Figures 2 and 5, with very diverse results and in some cases, non-convergence to a solution. This undoubtedly entails computing multiple times or making a systematic exploration with different parameters in the same environment. In this work, a total training of 42,000 agents was carried out, and 21,000,000 episodes with Q-Learning; before 2,000 agents and 1,000,000 M-Learning episodes.

For the Frozen Lake environment deterministic, the algorithm M-Learning shows faster learning with less variability in the results obtained by the trained agents compared to Q-Learning as presented in Figure 9. M-Learning takes 8 episodes less than Q-Learning for all agents to achieve score of 100 for the deterministic configuration of the environment. This represents a decrease of 30.7% in the number of episodes necessary for all agents to achieve a score of 100. Regarding the way in which the results obtained are distributed, Figure 10, although the average number of episodes to achieve the objective among the two algorithms have a difference close to one episode; M-learning presents a reduction of 58.37% in the standard deviation of episodes, which shows more consistent results.

For the stochastic Frozen Lake environment, the algorithm M-Learning shows faster learning with less variability in the results obtained by the trained agents compared to Q-Learning as presented in Figure 11. M-Learning takes 74 episodes less than Q-Learning to obtain a score above 70 for all agents in the stochastic configuration of the atmosphere. This represents a decrease of 61.66% in the number of episodes necessary for all agents to exceed a score of 70. Regarding how the results obtained are distributed, in Figure 12, the average number of episodes to achieve the objective is 45.84%. lower for M-Learning, similar to the deterministic environment M-Learning presents a reduction of 49.75% in the standard deviation of episodes, which shows more consistent results.

Finally, for the Frozen Lake environment in its two configurations and with delayed reward, the algorithm proposed M-Learning with a single parameter to tune Q_{lim}, presents better performance with a smaller number of episodes and less variability in the scores of the agents. The results obtained in this document show that the proposed heuristic for the propagation of the information with delayed reward is favorable for environments like Frozen Lake.

With the results obtained for M-Learning, it is proposed That future research continue with the evaluation of its behavior in other environments, to identify the problems where it can be useful.

References

1. Cottier, B.; Rahman, R.; Fattorini, L.; Maslej, N.; Owen, D. The rising costs of training frontier AI models, 2024, [arXiv:cs.CY/2405.21015].

 doi:10.20944/preprints202407.2253.v1

2. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning, 2013, [arXiv:cs.LG/1312.5602].
3. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. https://doi.org/10.1038/nature14236.
4. Sadhu, A.K.; Konar, A. Improving the speed of convergence of multi-agent Q-learning for cooperative task-planning by a robot-team. *Robotics and Autonomous Systems* **2017**, *92*, 66–80. https://doi.org/10.1016/j.robot.2017.03.003.
5. Canese, L.; Cardarilli, G.C.; Dehghan Pir, M.M.; Di Nunzio, L.; Spanò, S. Design and Development of Multi-Agent Reinforcement Learning Intelligence on the Robotarium Platform for Embedded System Applications. *Electronics* **2024**, *13*. https://doi.org/10.3390/electronics13101819.
6. Torres, J. *Introducción al aprendizaje por refuerzo profundo: Teoría y práctica en Python*; Direct Publishing, Independently Published, 2021.
7. Lapan, M. *Deep Reinforcement Learning Hands-On*; Packt Publishing: Birmingham, UK, 2018.
8. Balaji, N.; Kiefer, S.; Novotný, P.; Pérez, G.A.; Shirmohammadi, M. On the Complexity of Value Iteration, 2019, [arXiv:cs.FL/1807.04920].
9. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, second ed.; The MIT Press, 2018.
10. Jang, B.; Kim, M.; Harerimana, G.; Kim, J.W. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access* **2019**, *7*, 133653–133667. https://doi.org/10.1109/ACCESS.2019.2941229.
11. Liu, S.; Hu, X.; Dong, K. Adaptive Double Fuzzy Systems Based Q-Learning for Pursuit-Evasion Game. *IFAC-PapersOnLine* **2022**, *55*, 251–256. 16th IFAC Symposium on Large Scale Complex Systems: Theory and Applications LSS 2022, https://doi.org/10.1016/j.ifacol.2022.05.044.
12. Silva Junior, A.G.d.; Santos, D.H.d.; Negreiros, A.P.F.d.; Silva, J.M.V.B.d.S.; Gonçalves, L.M.G. High-Level Path Planning for an Autonomous Sailboat Robot Using Q-Learning. *Sensors* **2020**, *20*. https://doi.org/10.3390/s20061550.
13. Çimen, M.E.; Garip, Z.; Yalçın, Y.; Kutlu, M.; Boz, A.F. Self Adaptive Methods for Learning Rate Parameter of Q-Learning Algorithm. *Journal of Intelligent Systems: Theory and Applications* **2023**, *6*, 191–198. https://doi.org/10.38016/jista.1250782.
14. Zhang, L.; Tang, L.; Zhang, S.; Wang, Z.; Shen, X.; Zhang, Z. A Self-Adaptive Reinforcement-Exploration Q-Learning Algorithm. *Symmetry* **2021**, *13*. https://doi.org/10.3390/sym13061057.
15. Huang, J.; Zhang, Z.; Ruan, X. An Improved Dyna-Q Algorithm Inspired by the Forward Prediction Mechanism in the Rat Brain for Mobile Robot Path Planning. *Biomimetics* **2024**, *9*. https://doi.org/10.3390/biomimetics9060315.
16. Xu, S.; Gu, Y.; Li, X.; Chen, C.; Hu, Y.; Sang, Y.; Jiang, W. Indoor Emergency Path Planning Based on the Q-Learning Optimization Algorithm. *ISPRS International Journal of Geo-Information* **2022**, *11*. https://doi.org/10.3390/ijgi11010066.
17. dos Santos Mignon, A.; de Azevedo da Rocha, R.L. An Adaptive Implementation of ϵ-Greedy in Reinforcement Learning. *Procedia Computer Science* **2017**, *109*, 1146–1151. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal, https://doi.org/10.1016/j.procs.2017.05.431.
18. Zhang, M.; Cai, W.; Pang, L. Predator-Prey Reward Based Q-Learning Coverage Path Planning for Mobile Robot. *IEEE Access* **2023**, *11*, 29673–29683. https://doi.org/10.1109/ACCESS.2023.3255007.
19. Jin, W.; Gu, R.; Ji, Y. Reward Function Learning for Q-learning-Based Geographic Routing Protocol. *IEEE Communications Letters* **2019**, *23*, 1236–1239. https://doi.org/10.1109/LCOMM.2019.2913360.
20. Ou, X.; Chang, Q.; Chakraborty, N. Simulation study on reward function of reinforcement learning in gantry work cell scheduling. *Journal of Manufacturing Systems* **2019**, *50*, 1–8. https://doi.org/10.1016/j.jmsy.2018.11.005.
21. Li, Y.; Wang, H.; Fan, J.; Geng, Y. A novel Q-learning algorithm based on improved whale optimization algorithm for path planning. *PLOS ONE* **2022**, *17*, e0279438. https://doi.org/10.1371/journal.pone.0279438.
22. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Advances in Engineering Software* **2016**, *95*, 51–67. https://doi.org/10.1016/j.advengsoft.2016.01.008.

 doi:10.20944/preprints202407.2253.v1

23. Sowerby, H.; Zhou, Z.H.; Littman, M.L. Designing Rewards for Fast Learning. *ArXiv* **2022**, *abs/2205.15400*.
24. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym, 2016, [arXiv:id=1606.01540].