No. 134/25, 71–85
ISSN 2657-6988 (online)
ISSN 2657-5841 (printed)
DOI: 10.26408/134.05

Submitted: 17.04.2025
Accepted: 22.05.2025
Published: 18.06.2025

SELECTED REINFORCEMENT LEARNING METHODS APPLIED TO DETERMINE THE OPTIMAL PATH OF TRANSITION

Adrian Sawicki[1]*, Mirosław Tomera[2]

[1] Gdynia Maritime University Doctoral School, 81–87 Morska St., 81-225 Gdynia, Poland, ORCID 0009-0006-7904-4410, e-mail: a.sawicki@sd.umg.edu.pl

[2] Gdynia Maritime University, 81–87 Morska St., 81-225 Gdynia, Poland, Department of Ship and Industrial Automation, Faculty of Electrical Engineering, ORCID 0000-0003-4745-597X

*Corresponding author

Abstract: The focus of this work is the determination of the optimal path for a mobile agent to take in an environment with static obstacles, using reinforcement learning (RL). The paper explains the work examining different RL algorithms, such as *Q*-learning and Sarsa in the classic version and enhanced with the Adam gradient optimiser. The work investigates the impact of the Adam gradient optimiser on the rate and stability of finding the optimal solution. The analysis includes a comparison of the learning rate, the number of steps in a single episode and the stability of the learning process. The results reveal that the considered Q-learning and Sarsa algorithms supplemented with the Adam optimiser achieve a higher performance, characterised by a faster determination of the optimal transition path, than the same algorithms without the Adam gradient optimiser. The results could be particularly useful in practical applications for routing transitions in fields like mobile robotics.

Keywords: Artificial Intelligence, reinforcement learning, Q-learning, Sarsa, Adam optimizer.

1. INTRODUCTION

Reinforcement learning (RL) is one of the key issues in machine learning, enabling a computational agent to make autonomous decisions based on interactions with a dynamic environment. For a comprehensive discussion of RL, see Sutton and Barto [2018].

RLs find applications in solving different types of problems, most commonly for optimising the path of transition in navigation systems for mobile agents, as well as for mobile agent control [Cao et al. 2024]. Planning a transition path for an agent in an environment with static obstacles is a significant problem which requires an effective approach. Traditional path planning methods developed over the years, such as Dijkstra's *algorithm or A**, are insufficient as they lack flexibility when

encountering different changing environmental conditions and dynamically emerging obstacles [Tang and Ma 2021].

More recently, solutions have been emerging in which RL is used to determine the optimal transition paths, which allows the agent to autonomously adapt its movement strategy by learning optimal decisions in real time and in an unfamiliar environment [Adhirai and Kumar 2022]. For example, Abdalmanan et al. [2023] used reinforcement learning methods to determine a path of transition for a mobile agent in an unknown environment, based on a 2D LiDAR sensor. In this case, successful experiments were conducted to train the agent to reach a single destination and multiple destinations.

In autonomous robotics, an important issue is the navigation of mobile robots in unfamiliar environments in the presence of static obstacles and dynamic obstacles. The following papers have been written on this topic: [Viet, Kyaw and Chung 2011; Konar et al. 2013; Alhawary 2018; Sichkar 2019; Garaffa et al. 2021; Lee and Jeong 2021; Sachin et al. 2022; Song and Li 2023]. These algorithms are based on a raster grid, which is used to discretise the solution search space.

In this work, two selected RL algorithms, known as *Q*-learning and Sarsa, were investigated in their classical version and after being modified by using stochastic gradient optimisation, known as Adam [Kingma and Ba 2014].

2. REINFORCEMENT LEARNING (RL)

RL is an approach that enables an agent to make autonomous decisions in a changing environment, based on interactions with the environment and a reward system. A key aspect of RL is that the agent learns good behaviour, which, in the context of optimising the agent's path, allows it to find the optimal path for a transition that allows the agent to avoid obstacles and minimise the length of the path taken.

The basic RL elements are the computational agent and the environment, as shown in Figure 1. The task carried out in RL is to determine the agent's policy, denoted as $\pi(a_t|s_t)$. The agent, based of the received state value s_t, takes action a_t. The sequence of states, actions, and rewards are s_0, a_0, r_1, s_1, a_1, r_2, s_2,… etc. until a_{T-1}, r_T, s_T is a trajectory, denoted τ. The authors assumed that the control problem has a finite horizon which means that the decision-making task lasts for a fixed number of time steps, T, and ends with the last step.

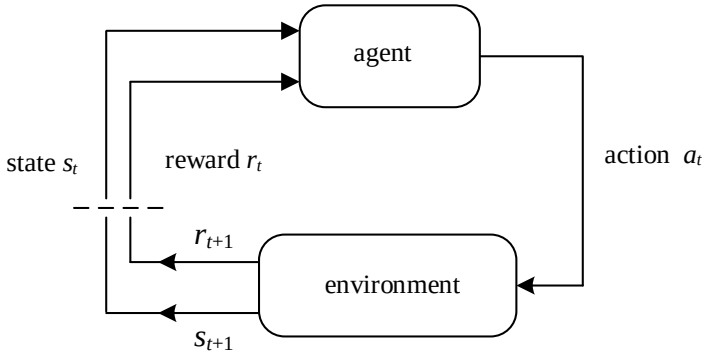

Fig. 1. The interaction of the agent AND the environment in RL

2.1. Markov property

For simplicity of consideration, RL was assumed to be modelled as a Markovian decision process. This allowed the application of the Markov property, which means that the future state of a system depends only on the present state and is independent of the past. This ensures that the decision-making by the agent does not require any analysis of its past history along the trajectory.
The Markovian decision process is a control process described by five variables: S, A, p, R, and γ [Bellman 1957]:

- S is a set of states (discrete or continuous);
- A is a set of agent actions (discrete or continuous);
- $p(s_{t+1}|s_t,a_t)$ is a transition function, specifying the probability of moving to state s_{t+1} after performing an action a_t in state s_t;
- $R(s_{t+1}|s_t,a_t)$ is a reward function assigning a value (reward or penalty) for the transition from s_t to s_{t+1} as a result of a_t;
- $\gamma \in [0,1]$ is a discount factor that determines the significance of future rewards relative to current rewards.

Figure 2 shows a graphical model of the Markovian decision process, including rewards, states and actions and transition probabilities, which are conditioned on both states and actions, hence $p(s_{t+1}|s_t, a_t)$.

2.2. Cumulative reward

Policy setting π is an optimisation process, where the function of destination:

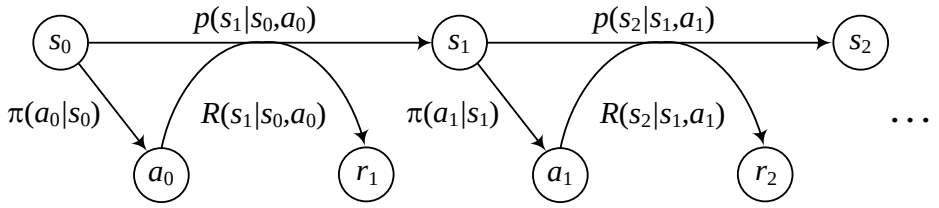

Fig. 2. Graphical model of the Markovian decision process

Source: [Francois-Lavet et al. 2018].

is the accumulated reward on trajectory $G(\tau)$

$$G(\tau) = \sum_{t=1}^{T} \gamma^t r_t \tag{1}$$

where discount factor $\gamma \in [0,1]$ reduces the weights as the value of time step t increases, while r_t is the immediate reward received at time step t. The value of reward r depends on the state and the action, or sometimes it can depend only on the state value and determines which states and actions are better. The objective of RL is to find parameters that define policy $\pi(a|s)$ and maximise the value of the total of rewards on trajectory $G(\tau)$.

2.3. Value function

The agent's objective is to find policy $\pi(a|s) \in \Pi$ that optimises the expected return $V^{\pi}(s)$, called the state value function, described by the following formula:

$$V^{\pi}(s) = E\left[\sum_{k=0}^{T} \gamma^k r_{t+k} \mid s_t = s, \pi\right] \tag{2}$$

From the definition of expected return (2), the optimal return can be easily determined as:

$$V^{*}(s) = \max_{\pi \in \Pi} V^{\pi}(s) \tag{3}$$

Another function considered in determining the optimal agent policy is the state-action value function, $Q^{\pi}(s,a)$, defined as:

$$Q^{\pi}(s,a) = E\left[\sum_{k=0}^{T} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right] \tag{4}$$

The equation above can be expressed recursively for a Markovian decision process, using the Bellman equation [Bellman and Dreyfuss 1962].

$$Q^{\pi}(s,a) = \sum_{s' \in S} p(s'\}s,a)\Big[R(s'|\, s,a) + \gamma Q^{\pi}(s', a = \pi(s'))\Big] \tag{5}$$

where *s'* is the state in the next time step. As for state value function *V*, the optimum value of state-action value function *Q* can be defined as:

$$Q^{*}(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a) \tag{6}$$

A particular feature of state-action value function *Q* compared to state value function *V* is that the optimal policy π^* can be obtained directly from $Q^*(s,a)$

$$\pi^{*}(s) = \arg\max_{a \in A} Q^{*}(s,a) \tag{7}$$

The optimal state value function $V^*(s)$ denotes the expected discounted reward when the agent is in a given state *s* and then applies the policy π^*. Optimal state-action value $Q^*(s, a)$ denotes the expected discounted return when the agent is in a given state *s* and, for a given action *a* in that state, further applies the policy π^* [*Francois-Lavet et al., 2018*].

3. RL ALGORITHMS

Popular value-based algorithms that build state-action value functions, which in turn allow policy determination, were selected for analysis. This work presents the most popular value-based algorithms which follow: *Q*-learning [Watkins 1989] and Sarsa [Rummery and Niranjan 1994], and their expansions using the Adam gradient optimiser [Kingma and Ba 2014].

3.1. Q-learning

Q-learning learns value function $Q(s,a)$, which is a prediction of the return associated with each action $a \in A$ in each state $s \in S$. A temporal difference (TD) is used to update state-action value function *Q*; the temporal difference is between the target value and the estimated value at different time steps [Sutton and Barto 1988]. The rudimentary method of determining TD is to update state value function $V(s)$ as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] \tag{8}$$

where α is the learning rate factor, while γ is the discount factor. In each state, state value function V corresponds to state-action value function Q with the action that has the highest predicted return, which can be expressed as:

$$V(s_t) \leftarrow Q(s_t, a_t) \tag{9}$$

Since the overriding objective is to maximise received return $Q(s_t,a_t)$ then state value estimation function $V(s_{t+1})$ can be expressed using the following formula:

$$V(s_{t+1}) \leftarrow \max_{a \in A} Q(s_{t+1}, a) \tag{10}$$

After substituting relations (9) and (10) into equation (8), the equation for one-step update in the Q-learning method is obtained:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{11}$$

As shown in Watkins [1989], this update converges to a certain fixed value for Markovian decision problems with a finite number of states, for which the Q-function is stored in an array. When convergence of the Q-function is obtained then the optimal policy is the action in each state with the highest projected return.

In Algorithm 1, there is a pseudo-code operating on equation (11), which allows the routing of a transition for a mobile agent using the Q-learning method.

Algorithm 1: Q-learning
Parameter values: $\gamma = 0.99$, $\alpha = 0.01$.

Initialise array $Q(s,a)$ for all $s \in S$, $a \in A$
repeat (for each episode):
  Initialise initial state s_0
  repeat (for each step t in the episode):
    $a_t \leftarrow \varepsilon\text{-greedy}(s_t, Q)$ (greed-based strategy)
    r_t, $s_{t+1} \leftarrow$ Environment (s_t, a_t)
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
    $s_t \leftarrow s_{t+1}$
  until final s
until stop

3.2. Sarsa

This method was proposed in Rummery and Niranjan [1994] and termed 'modified .*Q*-Learning'. This method is now called Sarsa, proposed by Richard Sutton [Sutton and Barto 2018**]** and derived from the first letters of the words (state-action-reward-state-action), a method used to determine array *Q* that holds the probability of executing action *a* in state *s*.

The rule of update in this method is formulated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t`} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{12}$$

Algorithm 2 shows the subsequent steps to update array *Q* based on equation (12).

Algorithm 2: Sarsa
Parameter values: $\gamma = 0.99$, $\alpha = 0.01$.

Initialise array $Q(s,a)$ for all $s \in S$, $a \in A$
repeat (for each episode):
  Initialise initial state s_0
  $a_0 \leftarrow \varepsilon\text{-}greedy(s_0, Q)$ (greed-based strategy)
  repeat (for each step t in the episode):
    $r_t, s_{t+1} \leftarrow$ Environment (s_t, a_t)
    $a_{t+1} \leftarrow \varepsilon\text{-}greedy(s_{t+1}, Q)$ (greed-based strategy)
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t`} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
    $s_t \leftarrow s_{t+1}$
    $a_t \leftarrow a_{t+1}$
  until final *s*
until stop

3.3. Q-learning I Sarsa with the Adam optimizer

The Adam optimiser (where Adam stands for 'Adaptive Moment Estimation') is an efficient stochastic optimisation algorithm that requires only first-order gradients with low memory requirements. The algorithm calculates the first and second moments of the gradients, which are estimates of the mean and uncentred variance of the gradients, respectively. These moments are used to update the model parameters [Kingma and Ba 2014].

Algorithm 3 features a pseudo-code showing the subsequent steps to update the parameters of array *Q* using *Q*-learning with the Adam optimiser. The algorithm

starts by initialising the first (m_t) and second (v_t) moment variables to zero. During each learning iteration, gradients of the model parameters against the loss function are calculated.

Algorithm 3: Q-learning with the Adam optimizer
Parameter values: $\gamma = 0.99$, $\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

$f(s,a;\theta) \leftarrow 0$ (initialisation of the evaluation function for all $s \in S$, $a \in A$)
$m_0 \leftarrow 0$ (initialisation of the first moment variable)
$v_0 \leftarrow 0$ (initialisation of second moment variable)
repeat (for each episode):
  Initialise initial state s_0
  Initialise time step, $t \leftarrow 0$
  repeat (for each step t in the episode):
    $Q(s,a) \leftarrow f(s,a;\theta_t)$
    $a_t \leftarrow \varepsilon\text{-greedy}(s_t, Q)$ (greed-based strategy)
    $r_t, s_{t+1} \leftarrow$ Environment (s_t, a_t) (agent's impact on the environment)
(a) $y \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a_t)$ (target function value)
    $\Delta Q \leftarrow Q(s_t, a_t) - y$
    $\nabla_\theta f(s,a;\theta) \leftarrow 0$ (initialisation of the gradient for all $s \in S$, $a \in A$)
    $\nabla_\theta f(s_t, a_t; \theta) \leftarrow \nabla_\theta f(s_t, a_t; \theta) + \Delta Q$
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f(s,a;\theta)$ (evaluation function gradient at step t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (first moment update)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (second moment update)
    $\hat{\alpha} \leftarrow \alpha \cdot \sqrt{1-\beta_2^t} \,/\, (1-\beta_1^t)$ (scaling of learning step size)
    $f(s,a;\theta_t) \leftarrow f(s,a;\theta_{t-1}) - \hat{\alpha} \cdot \left(m_t / (\sqrt{v_t} + \epsilon)\right)$ (parameter update)
(b) $s_t \leftarrow s_{t+1}$
  until final s
until stop

To obtain the pseudo-code for Sarsa with the Adam optimiser, algorithm 3 requires replacing the lines (a) and (b) with the following:

(a) $y \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$ (target function value)

(b) $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$

Apart from these two lines of code, there are no other differences in the pseudo-code for the Q-learning and Sarsa algorithms with the Adam optimiser.

4. SIMULATION STUDIES

This work focuses on the analysis of basic RL algorithms. Figure 3 shows a raster map containing the static obstacles and the determined path of transition using all four RL algorithms of interest. The set of possible actions of agent a was four-element, A = {N, S, E, W}, congruent with geographical directions (N – north, S – south, E – east, W – west).

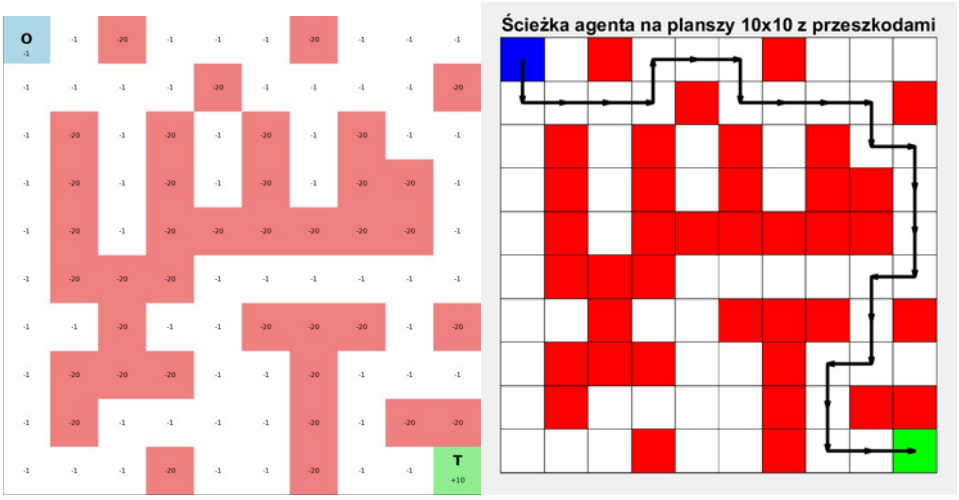

Fig. 3. Diagram of the test environment (left) and the agent's trajectory after the learning process end (right) produced for all four RL algorithms of interest

4.1. Result analysis

Table 1 shows selected parameters for evaluating the RL algorithms of interest. Based on the results in the table, there are distinct differences in the operation of the algorithm variants of interest. The least number of steps to reach the destination was shown by Q-learning with the Adam optimiser, which demonstrated the rapid progression to a stable policy. Sarsa with Adam achieved a very similar result, which may indicate that the Adam optimiser was working effectively in the early stages of training. The basic versions of both algorithms, Q-learning and Sarsa, achieved similar, significantly higher step count values, indicating slower learning rates.

Table 1. Summary of results for the selected assessment metrics

Algorithm	Basic Q-learning	Q-learning + Adam	Basic Sarsa	Sarsa + Adam
Total number of agent's steps	258721	140368	259678	141176
Average step count, success only	24.0667	24.0667	24.2667	24.2667
Average total step count	51.74	28.07	51.94	28.42
Step count standard deviation	63.95	27.07	64.47	29.09
Average reward	-13.67	-14.33	-13.27	-13.87
Average reward variance, last 1000	0.45	0.38	0.18	0.21
Average reward after 25% of training time	-191.2	-173.43	-192.94	-101.47
Average reward after 25% to 50% of training time	-46.94	-14.05	-47	-13.47
Improvement, % (1st stage vs 2nd stage)	75.4%	91.9%	75.6%	86.7%
Average final reward, last 1000	-13.43	-14.16	-13.34	-13.48

Considering the average number of steps (step count) per episode (which included failed episodes), Sarsa+Adam and Q-learning+Adam again performed better, with values of 28.42 and 28.07 respectively, demonstrating efficient finding of the optimal trajectories. The basic Sarsa and Q-learning versions required, on average, close to 52 steps per episode, which means significantly lower efficiency. In terms of stability (the step count standard deviation), the Adam-including algorithms were much more repeatable with deviations within 27 to 29, while the basic variants had more than 63.

Analysing the reward trend over time, the average reward after 25% of training time was weakest for the basic variants of Sarsa and Q-learning (-192.94 and -191.2), and significantly better for the Adam optimiser-enabled versions (-101.47 and -173.43). In the second stage (between 25% and 50% of episodes), Sarsa with the Adam optimiser and Q-learning with the Adam optimiser achieved very good values, -13.47 and -14.05, respectively. The improvement between these stages was the highest for Q-learning + Adam (91.9%) and Sarsa + Adam (86.7%), indicating a large increment in the quality of the strategy in a short time.

Comparing the average final award in the last 1000 episodes, the basic Q-learning (-13.43) performed best, followed closely by the basic Sarsa (-13.34); here, Sarsa + Adam was slightly weaker (-13.48). The weakest result was with Q-learning + Adam (-14.16).

4.2. Visualisation of the learning process dynamics

Figures 4–7 show the learning curves of the algorithms of interest. In these curves, the number of steps is shown on the vertical axis in a single episode, while the horizontal axis shows the number of each next episode. The maximum number of steps in a single episode was 500. This means that the agent should find the destination point on the map within this interval of steps; if the agent fails, the episode is interrupted and the next episode run from the starting point in the top left corner of the map.

The curves shown in Figures 4 to 7 provide a graphical assessment of the rate at which the optimum path of transition was found on the map shown in Figure 3. Sarsa + Adam converged very quickly to a low step count and a stable decision-making policy, confirming its high effectiveness early on in the training.

Significant improvements in strategy quality already occurred between the first and second stages of learning, which distinguishes this variant from the others. The basic Q-learning was characterised by the most even learning rate, and maintains a high stability of results with gradually improving quality of decisions. It was the most predictable and repeatable algorithm.

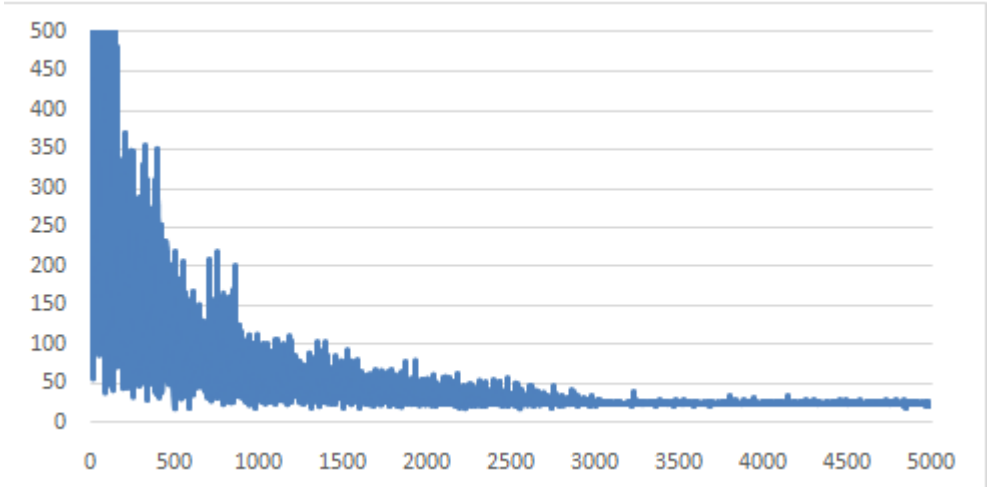

Fig. 4. Curve of the step count per episode for Q-learning (Algorithm 1)

In contrast, Q-learning + Adam achieved a significant improvement in the first stage of learning, but this did not translate into final quality. The large variance in results in the last episodes indicates instability, which could be problematic in applications that require consistent decisions. The basic Sarsa, despite the highest average overall award, had the slowest improvement dynamics and the highest step count, making it the least time-efficient algorithm.

All algorithms converged to comparable final trajectories, but differed in their style of arriving at optimal strategies, from rapid convergence and instability (Q-learning+Adam), high quality and predictability (Q-learning), to efficiency in learning at the expense of consistency (Sarsa + Adam).

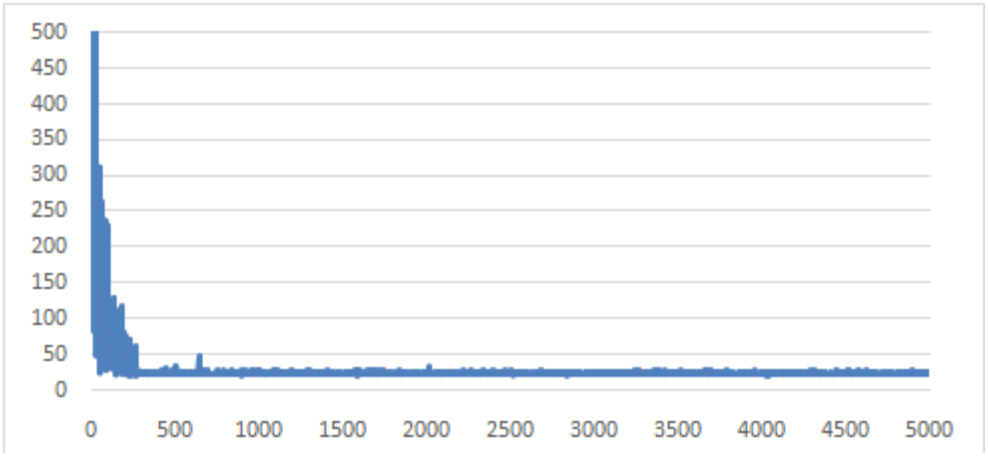

Fig. 5. Curve of the step count per episode for Q-learning + Adam (Algorithm 3)

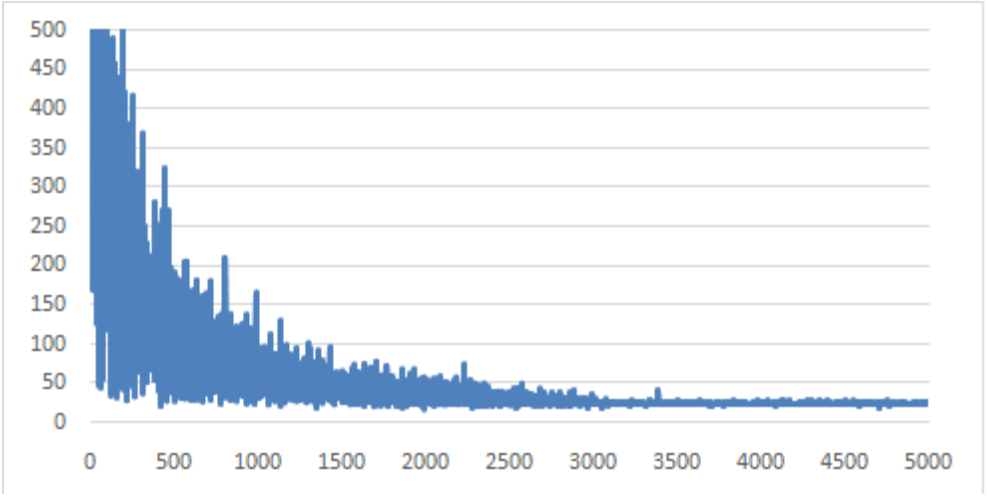

Fig. 6. Curve of the step count per episode for Sarsa (Algorithm 2)

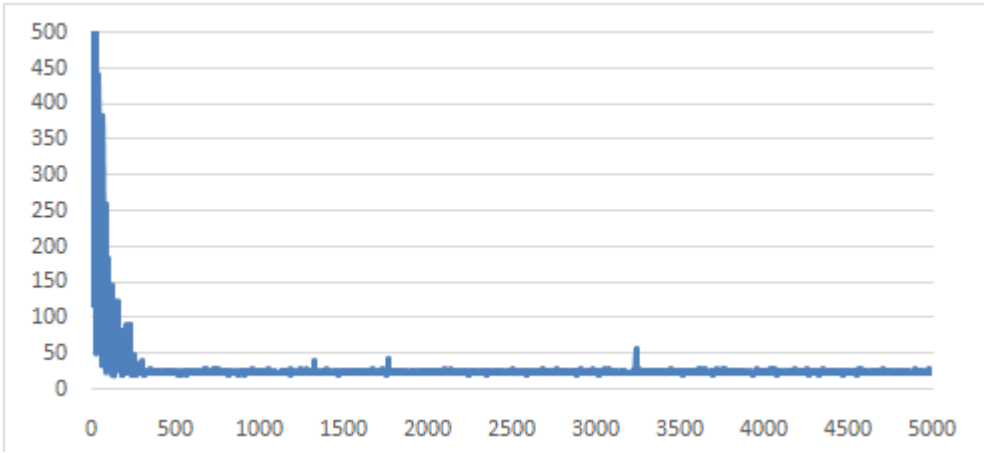

Fig. 7. Curve of the step count per episode for Sarsa + Adam (Algorithm 3 with code lines (a) and (b))

4.3. Evaluation in the context of practical applications

The (basic) Q-learning algorithm is recommended for environments where a balance between exploration and stability is important, e.g. in critical decision-making systems;
Sarsa with the Adam optimiser will perform well where fast learning and error reduction are a priority, such as in the navigation of autonomous vehicles.

The basic Sarsa algorithm can be useful in scenarios where the quality of decisions is more important than the rate at which they are reached, while the Q-learning with the Adam optimiser can be considered for tasks that require intensive exploration but have lower stability requirements.

5. CONCLUSIONS

The results show that there is no single algorithm for RL that is optimal in all aspects. Sarsa with the Adam optimiser achieved the best results in terms of convergence speed, number of steps taken and dynamic strategy improvement in the early phase of training. It can be a good choice for applications requiring quick performance with acceptable stability.

In contrast, the basic Q-learning stands out for having the highest quality final strategy and the greatest stability of results, both in terms of the number of steps and the reward. This makes it suitable for environments which require predictable and consistent decisions in the long term.

The two other variants, the basic Sarsa and Q-learning with Adam optimiser, presented clear limitations. The basic Sarsa learned too slowly and required more steps to reach the destination, despite a high average overall reward. In contrast, Q-learning with the Adam optimiser was characterised by high variability and poorer final results, despite a fast start and significant improvement in the initial stages.

The choice of algorithm should depend on the characteristics of the task and the application priorities: whether the rate of learning, the final quality of the strategy, or its stability and repeatability of performance are more important.

Future work include plans to expand the analysis to include environments with dynamic obstacles and tests under variable topology of space. This will allow a better assessment of the adaptability and robustness of the tested methods under real-world conditions.

REFERENCES

Adhirai, N., Kumar, S., 2022, *Reinforcement Learning Based Path Planning for Mobile Robots Traversing in Unknown Environments*, 6th National Conference on Recent Trends in Instrumentation and Control (RTIC 2022), Anna University, Chennai, India, pp. 101–106.

Alhawary, M., 2018, *Reinforcement – Learning – Based Navigation for Autonomous Mobile Robots in Unknown Environments*, Robotics and Mechatronics, University of Twente, Enschede, The Netherlands.

Bellman, R., 1957, *A Markovian Decision Process*, Journal of Mathematics and Mechanics, vol. 6, pp. 679–684.

Bellman, R.E., Dreyfus, S.E., 1962, *Applied Dynamic Programming*, Princeton University Press, Princeton, New Jersey, USA.

Cao, Y., Ni K., Kawaguchi, T., Hashimoto, S., 2024, *Path Following for Autonomous Mobile Robots with Deep Reinforcement Learning*, Sensors, vol. 24, no. 2, pp. 1–22.

Francois-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau J., 2018, *An Introduction to Deep Reinforcement Learning*, Foundations and Trends in Machine Learning, vol. 11, no. 3–4, pp. 219–354.

Garaffa, C., Basso, M., Konzen, A.A., de Freitas, E.P., 2021, *Reinforcement Learning for Mobile Robotics Exploration: A Survey*, IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 8, pp. 3796–3810.

Kingma, D.P., Ba, J.L. 2014, *Adam: A Method for Stochastic Optimization*, Computing Research Repository.

Konar, A., Chakraborty, I.G., Singh, S.J., Jain, L.C., Nagar, A.K., 2013, *A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot*, IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 43, no. 5, pp. 1141–1153.

Lee, H., Jeong, J., 2021, *Mobile Robot Path Optimization Technique Based on Reinforcement Learning Algorithm in Warehouse Environment*, Applied Sciences, vol. 11, no. 3, pp. 1–16.

Rummery, G.A., Niranjan, M., 1994. *On-line Q-learning Using Connectionist Systems*, Technical Report, Cambridge University Engineering Department, Cambridge, UK.

Sachin, V., Hashir, A.K., Mohammed, S.O., Vishnu, R., Syed, M.F., Imthias, A.T.P., 2022, *Motion Planning and Obstacle Avoidance of Mobile Robot in a Stochastic Environment Using Reinforcement Learning*, Proceedings of the International Conference on Aerospace and Mechanical Engineering (ICAME), pp. 1–6.

Sichkar, V.N., 2019, *Reinforcement Learning Algorithms in Global Path Planning for Mobile Robot*, Proceedings of the 2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), pp. 1–5.

Song, L., Li, D.Z., 2023, *Local Path Planning Via Improved Fuzzy and Q(λ)-Learning Algorithms for the Mobile Robot*, Journal of Computers, vol. 34, no. 5, pp. 265–284.

Sutton, R.S., Barto, A.G., 2018, *Reinforcement Learning: An Introduction*, Adaptive Computation and Machine Learning, MIT Press, Cambridge Massachusetts, USA.

Tang, Z., Ma, H., 2021, *An Overview of Path Planning Algorithms*, IOP Conference Series: Earth and Environmental Science, vol. 804, pp. 1–10.

Viet, H.H., Kyaw, P.H., Chung, T.C., 2011, *Simulation-Based Evaluations of Reinforcement Learning Algorithms for Autonomous Mobile Robot Path Planning*, [in:] Park, J.J., Arabnia, H., Chang, H.B., Shon, T. (eds.), *IT Convergence and Services*, Lecture Notes in Electrical Engineering, vol. 108, Springer, pp. 467–476.

Watkins, C., 1989, *Learning from Delayed Rewards*, Ph.D. dissertation, Cambridge University, Cambridge, UK.