DOI: 10.2507/37th.daaam.proceedings.xxx

# DESIGN AND ANALYSIS OF SEARCH AND REINFORCEMENT LEARNING ALGORITHMS IN THE MAZE NAVIGATION PROBLEM

Marijana Bandic & Kresimir Rakic

Working Paper of 37th DAAAM Symposium

**This Publication has to be referred as:** Bandic, M[arijana] & Rakic, K[resimir] (2026). Design and Analysis of Search and Reinforcement Learning Algorithms in the Maze Navigation Problem, Proceedings of the 37th DAAAM International Symposium, pp.xxxx-xxxx, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-xx-x, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/37th.daaam.proceedings.xxx

**Abstract**

This paper deals with the design, implementation, and experimental analysis of algorithms for maze navigation. Algorithms based on breadth-first search (BFS), depth-first search (DFS), the A* algorithm and Q-learning were developed. Their efficiency was evaluated through multiple tests on perfect and imperfect mazes of different dimensions. The comparison was made according to path length, number of explored nodes, execution time and memory usage. Special attention was given to the convergence of the Q-learning agent and the impact of different parameters on learning quality. The results show that BFS and A* most often find the shortest paths, DFS uses less memory, while Q-learning can learn a good policy but requires sufficient training.

**Keywords:** maze navigation, pathfinding algorithms, A* search, Q-learning.

## 1. Introduction

Navigation tasks of robots, autonomous vehicles and video game characters share a common navigation problem: how to find a path from one point to another in a space whose structure is not known in advance and must be explored. A maze is a simplified representation of such a space. In computer science, the maze becomes a controlled space for thorough testing and comparison of navigation algorithms.

Although the problem of finding a path through a maze seems simple, it raises questions related to some of the most important ideas of artificial intelligence. On the one hand, there are classical algorithms for state-space search, such as depth-first search, breadth-first search and informed search such as the A* algorithm. These algorithms have a long history, solid theoretical foundations and clearly defined properties, which enables precise prediction of their behaviour. On the other hand, there is a fundamentally different approach based on reinforcement learning, where the algorithm does not know the structure of the environment in advance but must discover it by trial and error, relying on the rewards and penalties it receives during the interaction. One of the best known examples of this paradigm is the Q-learning algorithm. It does not require prior knowledge of the layout of the maze, it learns from its own experience, gradually developing an estimate of the value of each move.

Recent studies have examined maze navigation from both classical search and reinforcement learning perspectives. Elkari Eidia et al. compared DFS, BFS and A* in maze path planning, focusing on path cost and algorithmic complexity [1]. In reinforcement-learning-based navigation, Xu investigated Q-learning in grid environments and highlighted the influence of the exploration-exploitation trade-off on training [2]. Building on these studies, the present work compares classical graph-search algorithms and tabular Q-learning under a common experimental setup. The main research question is how classical graph-search algorithms and tabular reinforcement learning differ in solution quality, search effort, execution cost and scalability when applied to the same maze instances. In addition, the study investigates whether the performance of Q-learning can be improved through appropriate hyperparameter selection and whether its learned policy remains effective as the maze size increases.

## 2. Methodology

### *2.1. Maze model and system design*

The system is implemented in the Java programming language and uses a modular architecture in which each functional unit is isolated in a separate package [3]. The key decision in the implementation was to separate the maze model from the agents that solve it. The Maze class does not know how the maze is solved. Its only responsibility is to accurately model the state space and to enable queries about the available transitions from a given cell. The maze is implemented as a two-dimensional matrix of cells, while its functional representation corresponds to an undirected graph. Each cell is a node and reachable adjacent cells form edges. The maze does not store all edges explicitly, reachable neighbours are generated dynamically during execution, providing an implicit graph representation. The starting position is the upper-left cell and the goal position is the lower right cell.

Two maze generators were implemented. Perfect mazes are generated by randomized depth-first search using an explicit stack instead of recursive calls. The resulting maze contains exactly one path between any two cells and no cycles. Imperfect mazes are generated by first creating a perfect maze and then randomly removing additional walls. The default probability of removing an additional wall is 20%, which introduces cycles and multiple possible paths.

### *2.2. Navigation algorithms and Q-learning*

When choosing a search strategy, four properties are most commonly observed: completeness, optimality, time complexity and space complexity [4]. If no information is used when choosing the search strategy, the search is called uninformed, blind or brute-force search. On the other hand, if the choice uses some information which may or may not be reliable, the search is called informed or directed search [5].

The strategy followed by depth-first search is to search “deeper” in the graph whenever possible [6]. It uses a LIFO (Last-In-First-Out) principle, which means that the last discovered node is used for expansion, therefore it is implemented using stack [4]. The main feature of the DFS algorithm is low memory consumption. The algorithm does not take into account the length of the path found, so the first path found is not necessarily the shortest.

Unlike the DFS algorithm, BFS uses a queue that follows the FIFO (First-In-First-Out) principle. The node that was first added to the queue will be the first to be processed. In problems in which all transitions from one state to another have the same cost, as in a maze, BFS will find the shortest possible path [4].

A* algorithm belongs to the group of best-first search algorithms, where nodes with a higher estimated distance are temporarily deprioritized in favour of more promising ones [7]. The estimate is based on the evaluation function:

$$f(n) = g(n) + h(n) \tag{1}$$

Where g(n) is the cost of the path from the start node to node n, h(n) is the heuristic estimate of the remaining cost from node n to the goal [8][9]. Two heuristic functions were evaluated: Manhattan distance, which corresponds to movement in the four basic directions, and Euclidean distance, which represents the air-line distance between two points. Both heuristics are admissible for the considered four-directional grid because neither heuristic overestimates the actual shortest-path cost.

The time and space complexity of the implemented search algorithms are summarized in Table 1. The presented values represent the worst-case asymptotic complexity, where (b) denotes the branching factor, (d) the depth of the shallowest solution, and (m) the maximum depth of the search space.

| Algorithm | Time complexity | Space complexity |
|---|---|---|
| DFS | $O(b^m)$ | O(bm) |
| BFS | $O(b^d)$ | $O(b^d)$ |
| A* | $O(b^d)$ | $O(b^d)$ |

Table 1. Comparison of the time and space complexity of search algorithms [4]

Working Paper of 37th DAAAM Symposium

Q-learning is a model-free method [10], which means that the agent does not know how the environment works nor what consequences will follow after an action. The Q-table is a three-dimensional array indexed by row, column and direction. It stores the estimated Q value for each combination of state and action. The reward function assigns -5 for hitting a wall, +100 for reaching the goal, and -1 for an ordinary step. During training, actions are selected using an epsilon-greedy policy, in which the agent with probability ε randomly chooses an action for exploring the state space, while in the remaining cases it executes the action with the highest estimated value of the Q function. Q-values are updated after each action.

$$Q(s,a) = (1-\alpha) \times Q(s_t, a_t) + \alpha \times (r_t + \gamma \times \max_a Q(s_{t+1}, a)) \quad (2)$$

Where α is the learning rate coefficient, $r_t$ is the current reward, γ is the discount factor, $s_{t+1}$ is the new state after the executed action, and $\max_a Q(s_{t+1}, a)$ is the estimate of optimal future values [11].

### *2.3. Experimental design*

The scalability experiment tested maze dimensions of 10×10, 20×20, 30×30, 50×50, and 100×100 cells. For each dimension, 10 independent repetitions were performed. Within each repetition, all five agents solved the same maze instance, removing the effect of random differences in maze topology from the algorithm comparison. The experiment was conducted separately for perfect and imperfect mazes. For every agent execution the experiment recorded the algorithm name, maze dimensions, repetition number, whether a path was found, path length, number of explored nodes, execution time, and estimated memory consumption. In the scalability experiment, Q-learning was trained for 5000 episodes for each maze instance, and the maximum number of steps per episode was set to size×size×10. The reported execution time and memory usage for Q-learning represent post-training policy execution only and exclude the 5000 training episodes.

The Q-learning convergence experiment was conducted on a 30×30 maze. Each parameter configuration was trained for 1000 episodes and repeated five times. The tested learning rates were α = {0.1, 0.3, 0.5, 0.7}, discount factors were γ={0.8, 0.9, 0.99}, and epsilon-decay strategies were {0.99, 0.995, 0.999} with minimum ε=0.01. Average reward and the success rate of episodes in which the agent reached the goal were used as measures of learning performance. Execution time was measured using Java's System.nanoTime(), while memory usage was estimated from the difference between the JVM-reported used memory before and after execution.

All experiments were performed on the same laptop and under the same hardware and software conditions, in order to minimize the influence of external factors on the measured execution time and memory consumption.

## 3. Results

### *3.1. Comparison of search algorithms*

Before the analysis of the results, it is useful to show the behaviour of an individual algorithm on a concrete example. Fig. 1 shows a comparative view of solving the same imperfect maze of 10×10 dimensions by all five agents. The light blue cells represent cells visited by the agent, while the yellow cells represent the final path. It is immediately visible that BFS visits a larger set of cells, while DFS did not find the shortest path obtained by the other search algorithms. It should also be emphasised that the animation shown for the Q-learning agent shows only the steps in the solving phase, so exploration process is not visible.

Working Paper of 37th DAAAM Symposium

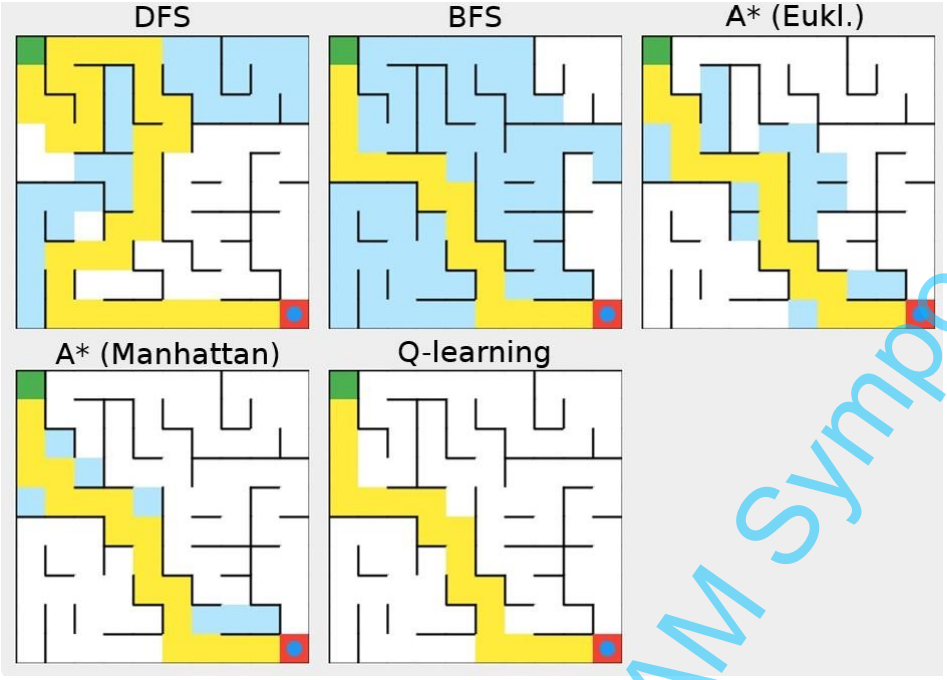


Fig. 1. Visual comparison of maze navigation algorithms

Fig. 2 shows an overview of the four primary metrics for all tested dimensions of the perfect maze. The graph in the upper left corner shows that all algorithms find the same path in perfect mazes, which is understandable since it is also the only path. These paths grow from about 30 steps in small mazes to almost 2050 steps in larger ones. Q-learning also reproduces the unique path on smaller perfect mazes, but its success rate decreases as the maze size increases, which means that it did not manage to learn the optimal path. Table 2 shows that the success rate of Q-learning was 60% for the 50×50 mazes and 0% for the 100×100 mazes. Although all algorithms find the same path, some explore fewer nodes than the others. BFS explores the most nodes, followed closely by A* (Euclidean), then A* (Manhattan), while DFS has fewer visited nodes. Of course, Q-learning explores the fewest nodes because the corresponding measure represents the number of movement steps executed during the learned-policy solve phase.

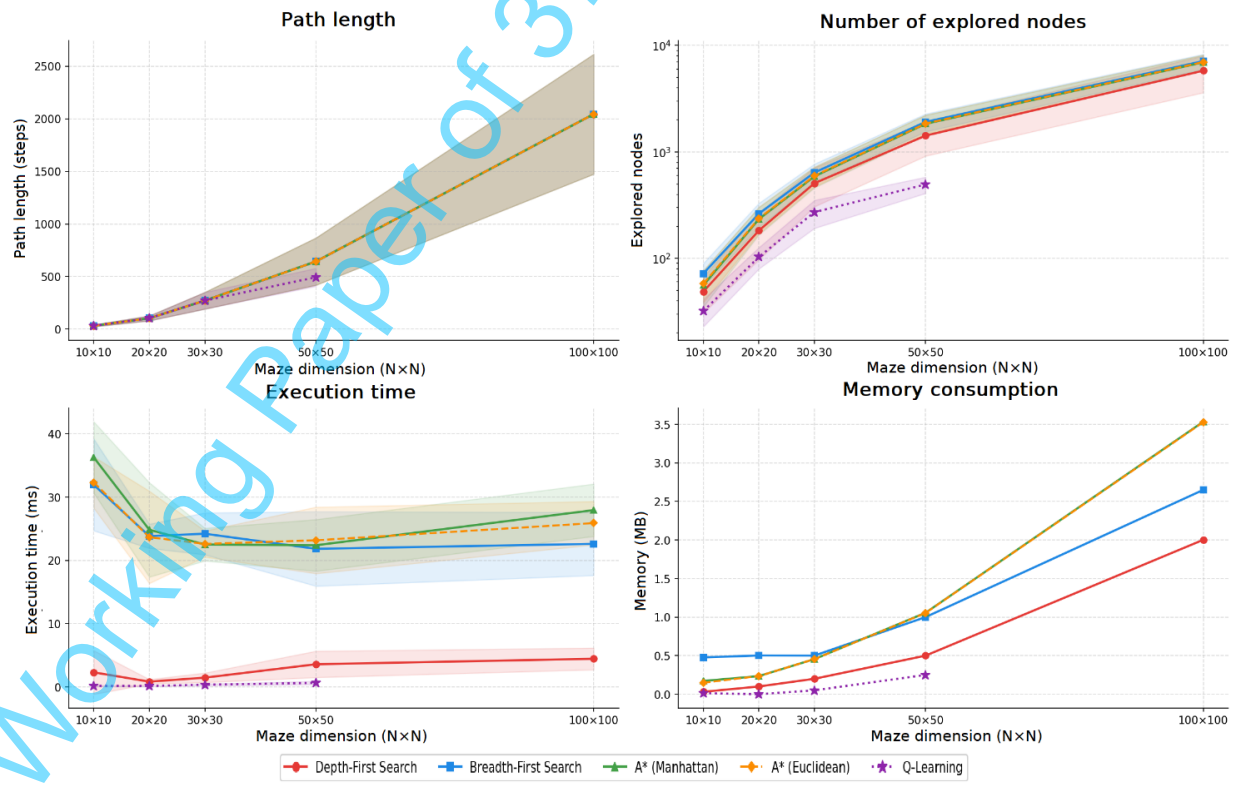


Fig. 2. Comparison of scalability metrics of navigation algorithm on perfect mazes

Working Paper of 37th DAAAM Symposium

With regard to execution time there is a large difference between algorithms. Table 2 shows that BFS, A* have similar execution times (20-40 ms) regardless of maze size, while for DFS and Q-learning this time is considerably lower. A clear difference can also be seen in memory consumption, where both A* algorithms consume the same amount of memory and, as maze dimensions increase, consume more than the others. DFS and Q-learning consume significantly less, with the note that Q-learning uses the learned policy, so its decision-making is reduced to a simple table lookup.

| Size | Algorithm | Success rate | Path length | Explored nodes | Time (ms) | Memory (B) |
|---|---|---|---|---|---|---|
| 10 | DFS | 100% | 32.80 | 48.50 | 2.31 | 32597 |
| 10 | BFS | 100% | 32.80 | 71.70 | 31.94 | 500692 |
| 10 | A*(Manhattan) | 100% | 32.80 | 55.70 | 36.27 | 180045 |
| 10 | A*(Euclidean) | 100% | 32.80 | 57.50 | 32.32 | 155584 |
| 10 | Q-learning | 100% | 32.80 | 31.80 | 0.17 | 14045 |
| 50 | DFS | 100% | 643.40 | 1416.30 | 3.59 | 524288 |
| 50 | BFS | 100% | 643.40 | 1906.00 | 21.83 | 1048576 |
| 50 | A*(Manhattan) | 100% | 643.40 | 1828.80 | 22.39 | 1107168 |
| 50 | A* (Euclidean) | 100% | 643.40 | 1841.50 | 23.18 | 1104216 |
| 50 | Q-learning | 60% | 493.33 | 492.33 | 0.65 | 262144 |
| 100 | DFS | 100% | 2044.00 | 5778.60 | 4.45 | 2099652 |
| 100 | BFS | 100% | 2044.00 | 7111.80 | 22.61 | 2778726 |
| 100 | A*(Manhattan) | 100% | 2044.00 | 6925.20 | 27.94 | 3705922 |
| 100 | A* (Euclidean) | 100% | 2044.00 | 6967.10 | 25.90 | 3699678 |
| 100 | Q-learning | 0% | - | - | - | - |

Table 2. Scalability metrics of navigation algorithms on perfect mazes

Fig. 3 shows the results for imperfect mazes with cycles. Unlike perfect mazes, differences in the quality of the paths found are noticeable here. BFS and both variants of the A* algorithm find the shortest paths, while DFS, due to its search strategy, often finds a considerably longer solution. Q-learning achieves near-optimal path lengths in most tested imperfect mazes. The differences are also visible in the number of explored nodes. BFS explores the most nodes in order to ensure the optimal solution. DFS explores fewer, but thereby sacrifices the quality of the found path. Both variants of A* lie between these two extremes, with the Manhattan heuristic showing somewhat better efficiency.

In execution time and memory consumption, the biggest change is seen in the A* algorithm with the Euclidean heuristic, which shows a longer execution time and higher memory consumption on larger mazes. Q-learning still has the shortest execution time, while DFS keeps low memory and time consumption in comparison with BFS and A*. It is important to note that Q-learning reached the goal for all tested imperfect-maze sizes under the selected training configuration, unlike in the case of perfect mazes. The additional passages introduced in imperfect mazes can create shorter routes to the goal, which may make successful trajectories easier to discover during training.

| Size | Algorithm | Success rate | Path length | Explored nodes | Time (ms) | Memory (B) |
|---|---|---|---|---|---|---|
| 10 | DFS | 100% | 29.20 | 41.20 | 2.39 | 1070 |
| 10 | BFS | 100% | 21.40 | 90.50 | 27.61 | 356292 |
| 10 | A*(Manhattan) | 100% | 21.40 | 51.10 | 27.98 | 206607 |
| 10 | A*(Euclidean) | 100% | 21.40 | 61.60 | 29.67 | 207696 |
| 10 | Q-learning | 100% | 21.40 | 21.40 | 0.09 | 4072 |
| 50 | DFS | 100% | 304.80 | 1070.80 | 3.20 | 524288 |
| 50 | BFS | 100% | 111.60 | 2493.70 | 23.56 | 1048576 |
| 50 | A*(Manhattan) | 100% | 111.60 | 1161.10 | 25.96 | 832434 |
| 50 | A* (Euclidean) | 100% | 111.60 | 1911.40 | 27.38 | 1240811 |
| 50 | Q-learning | 100% | 112.20 | 112.20 | 0.16 | 14050 |
| 100 | DFS | 100% | 709.40 | 3278.60 | 4.59 | 1363148 |
| 100 | BFS | 100% | 213.00 | 9993.80 | 31.05 | 3670016 |
| 100 | A*(Manhattan) | 100% | 213.00 | 4381.70 | 29.95 | 2738008 |
| 100 | A* (Euclidean) | 100% | 213.00 | 8095.70 | 40.67 | 4570593 |
| 100 | Q-learning | 100% | 213.60 | 213.60 | 0.32 | 262144 |

Table 3. Scalability metrics of navigation algorithms on imperfect mazes

Working Paper of 37th DAAAM Symposium

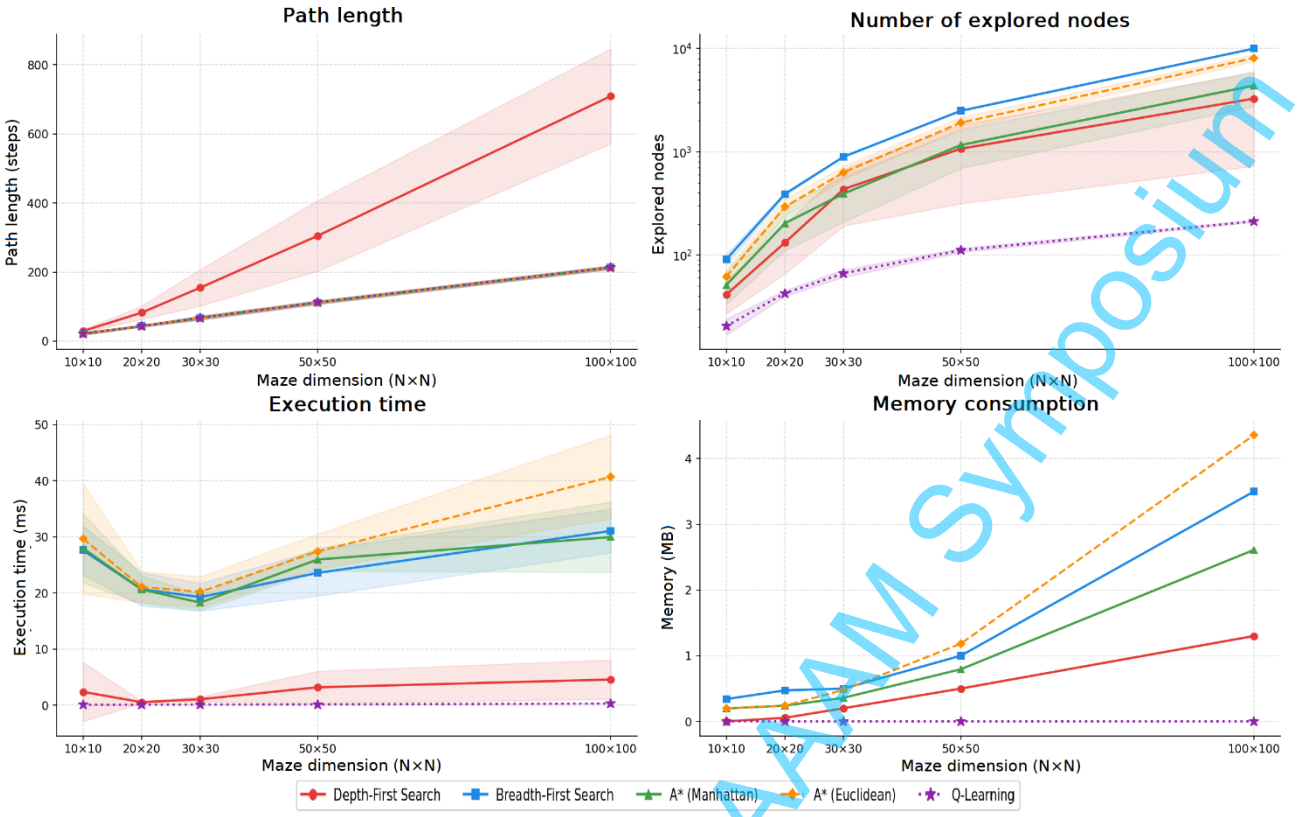


Fig. 3. Comparison of scalability metrics of navigation algorithm on imperfect mazes

### *3.2. Q-learning convergence analysis*

Before analysis, for easier understanding, Fig. 4 shows the learning progress on a 10×10 maze. The systematic hyperparameter analysis was performed separately on 30×30 mazes. The results after 0, 50, 100, 200 and 500 episodes of training are shown. The agent was trained on 1000 episodes. At the beginning, the agent knows nothing about the maze, so it behaves randomly and explores a large part of the space. Thus, in the first episode it made the maximum number of steps (1000). In earlier episodes the agent often makes cycles in moving between two or more positions. The path length in the 50$^{th}$ episode is 140, and after 100 episodes 68 steps. By episode 200, the trajectory length is reduced to 54 steps. After 500 episodes, the agent finds a path of 30 steps in length and moves directly towards the goal without unnecessary deviations. Since further training up to 1000 episodes did not bring visible changes, it can be concluded that the agent already achieved a stable policy at 500 episodes on this maze.

Working Paper of 37th DAAAM Symposium

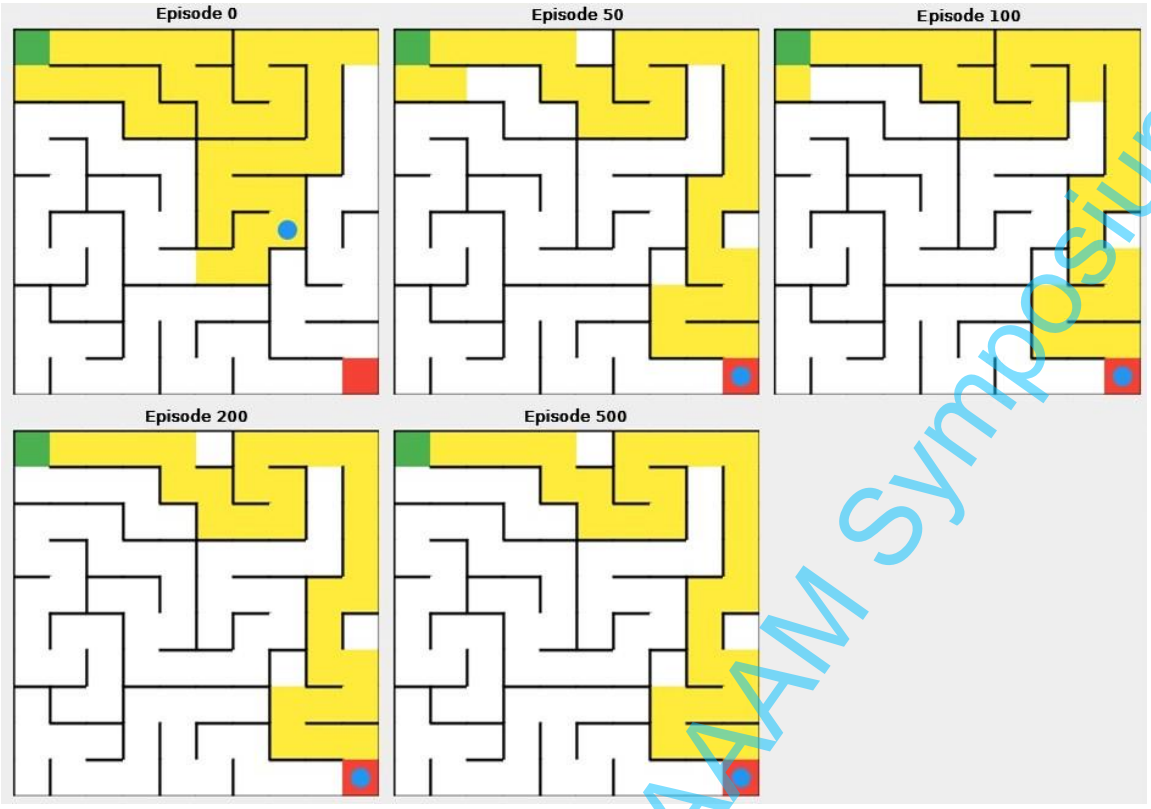


Fig. 4. Visual representation of the progress of the Q-learning agent during training

Table 4 and Fig. 5 show how different values of the parameter α affect the convergence of the algorithm. At the beginning of training all configurations achieve similar average rewards because the agent has no knowledge of the environment. The differences are noticed only after approximately 600 episodes. The best results are achieved by the configuration α=0.7, which achieves the highest average reward and progresses fastest towards a stable policy. These results show that a higher value of the parameter α enables a more efficient use of new experiences.

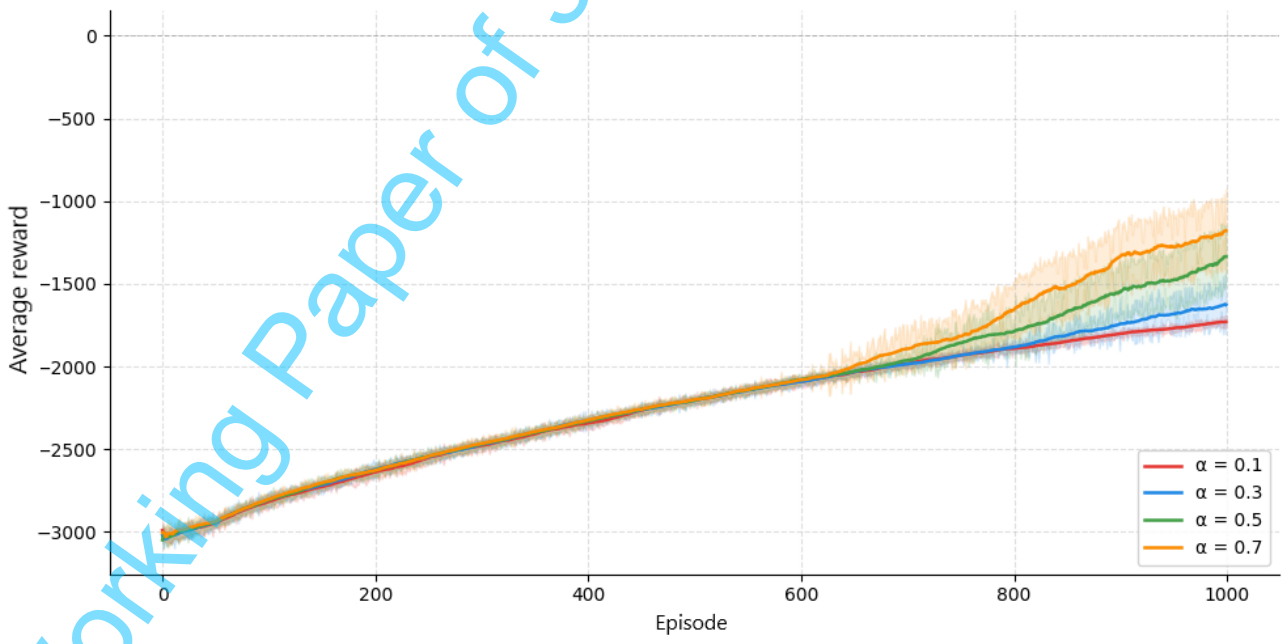


Fig. 5. Influence of learning rate (α) on convergence, (γ=0.9, ε-decay=0.999)

Working Paper of 37th DAAAM Symposium

| Episode \ α | 0.1 | 0.3 | 0.5 | 0.7 |
|---|---|---|---|---|
| 0 | -2990 | -3020 | -3048 | -3004 |
| 200 | -2646 | -2630 | -2567 | -2637 |
| 400 | -2280 | -2336 | -2340 | -2269 |
| 600 | -2066 | -2066 | -2044 | -2037 |
| 800 | -1868 | -1894 | -1880 | -1588 |
| 999 | -1723 | -1741 | -1361 | -1228 |

Table 4. Influence of the learning rate α over episodes (γ=0.9, ε-decay=0.999)

Fig. 6 shows a comparison of three strategies of decreasing the parameter ε. The rate of decreasing exploration has significant influence on the learning process. The strategy of fast decrease of ε (0.990) transitions early from exploration to the use of learned knowledge. Thus it achieves the fastest increase in average reward but it may be too aggressive. The medium strategy (0.995) behaves similarly, only at a slightly slower pace. On the other hand, the slow strategy (0.999) retains a high degree of exploration during a larger part of training and improves its reward result more slowly. Table 5 shows the concrete values of average reward depending on the decay strategy across different episodes.

| Decay \ Episode | 0 | 200 | 400 | 600 | 800 | 999 |
|---|---|---|---|---|---|---|
| Fast decay | -2994 | -1317 | -1093 | -1044 | -639 | -522 |
| Medium decay | -3062 | -1693 | -1256 | -1016 | -767 | -688 |
| Slow decay | -3020 | -2630 | -2336 | -2066 | -1894 | -1741 |

Table 5. Influence of the ε-decay strategy over episodes (α =0.3, γ=0.9)

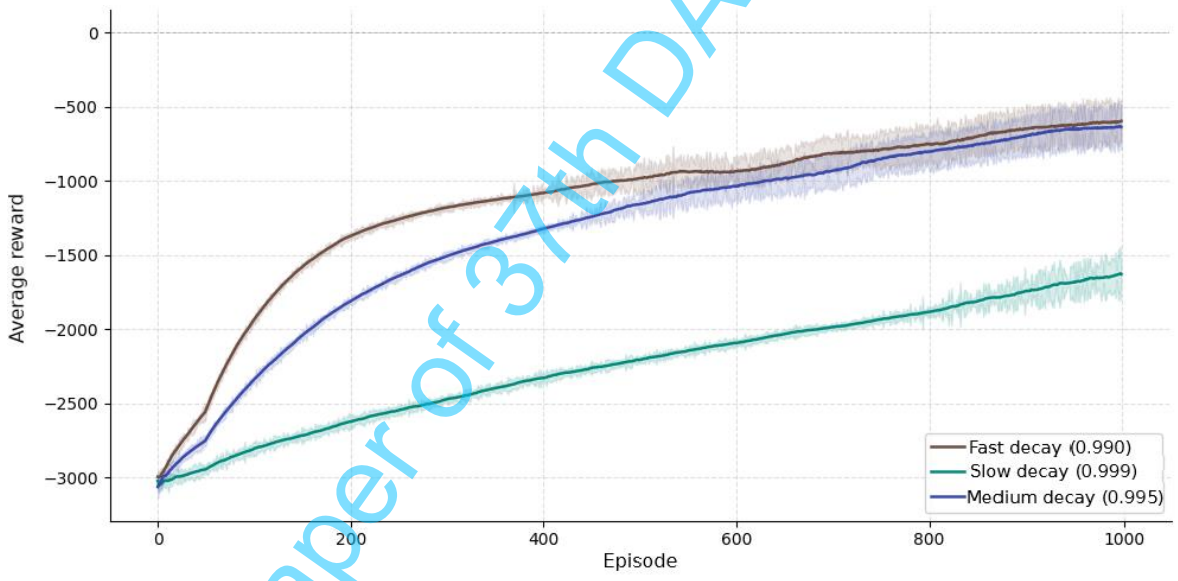


Fig. 6. Influence of the ε-decay strategy over episodes (α =0.3, γ=0.9)

Fig. 7 shows the change in the rate of successful episodes during training for different combination of the parameters α, γ and the ε-decay strategy. It is visible that the choice of parameters significantly affects the ability of the agent to reach the goal. With fast ε-decay, the highest-valued configurations achieve a high success rate already before the end of training. Combinations with high values of α and γ reach almost 100% of successful episodes. In Table 6 it can be noticed that a part of the configurations with lower parameter values remains at a low level and does not converge. With slow ε-decay, it is visible that no combination of parameters enables a hundred percent success rate. Combinations with high parameter values achieve a success rate of up to 71%. Medium decay represents a compromise between these two approaches.

Working Paper of 37th DAAAM Symposium

| Decay \ α, γ | All combination with α=0.1 | α=0.3 γ=0.8 | α=0.3 γ=0.9 | α=0.3 γ=0.99 | α=0.5 γ=0.8 | α=0.5 γ=0.9 | α=0.5 γ=0.99 | α=0.7 γ=0.8 | α=0.7 γ=0.9 | α=0.7 γ=0.99 |
|---|---|---|---|---|---|---|---|---|---|---|
| Fast decay | 0% | 0% | 73% | 71% | 0% | 100% | 100% | 0% | 100% | 100% |
| Medium decay | 0% | 0% | 71% | 69% | 0% | 100% | 100% | 0% | 100% | 100% |
| Slow decay | 0% | 0% | 20% | 24% | 0% | 51% | 53% | 0% | 68% | 71% |

Table 6. Success rate averaged over the final 100 training episodes for different α and γ combinations

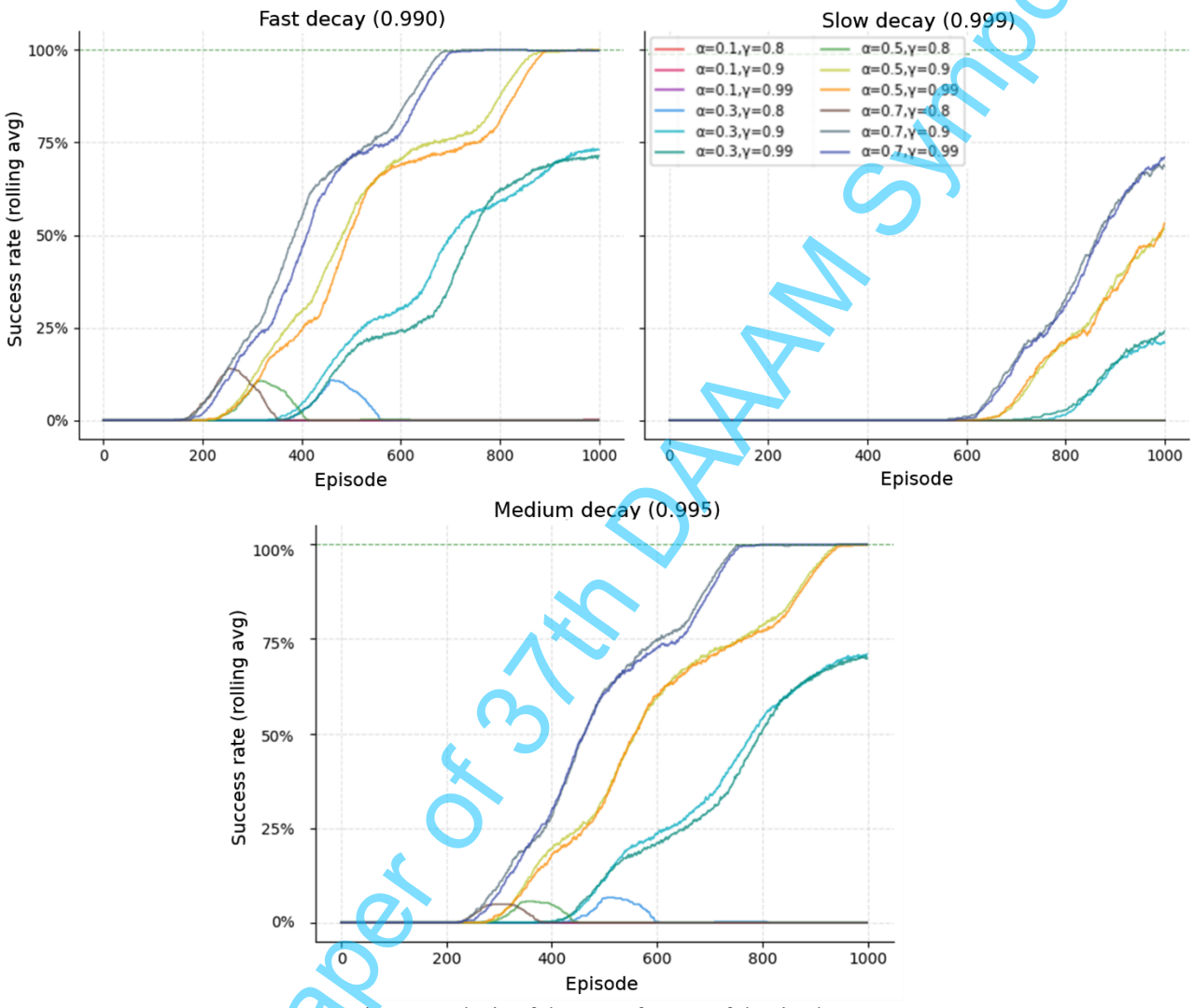


Fig. 7. Analysis of the rate of successful episodes

It can be noticed that all tested combinations that include the value α=0.1 have success rate of 0% in all examples. The behaviour of the configurations with discount factor γ=0.8 is also visible and interesting. During the early phases of learning with fast and medium decay, it is visible that the success rate reaches about 13%, after which it gradually declines and finally converges to 0%.

## 4. Discussion

The choice of the best algorithm for navigation through a maze depends greatly on what is actually meant by the word "best". The conducted experimental evaluation confirmed the theoretical expectations, while at the same time revealing nuances that cannot be concluded from the theoretical analysis of algorithm complexity.

Breadth-first search and the A* algorithm proved to be a reliable choice when path optimality is crucial, finding the shortest solution in mazes with cycles. Depth-first search, although it is the most memory-efficient, sometimes pays the price of considerably longer paths.

In the case of finding a path through a maze, the effectiveness of the Q-learning algorithm depends on the size of the state space. In smaller mazes the number of possible states and actions is relatively small, so during training the agent can

Working Paper of 37th DAAAM Symposium

visit a large part of the state space and successfully learn the optimal movement policy. However, as the maze dimensions increase, the number of states grows, which significantly increases the size of the Q-table and the number of required interactions with the environment. In very large mazes the agent often fails to visit all relevant states enough times within the given number of episodes. Because of this, some values in the Q-table remain insufficiently updated and the agent fails to find the goal. Tabular Q-learning is well suited to smaller discrete state spaces, where the state-action space can be sufficiently explored during training. As the state space grows, the fixed tabular representation becomes increasingly difficult to train within a limited number of episodes.

The behaviour of the configurations with the discount factor $\gamma$=0.8 indicates that in the initial episodes the agent randomly finds the goal thanks to the exploration of the state space, but due to the low value of $\gamma$ it insufficiently values long-term rewards. At the same time, the decrease of $\varepsilon$ reduces the level of exploration before a quality policy has been learned, due to which the agent prematurely begins to exploit poor solutions. The presented measurements have limitations. The memory measurement is not absolutely precise due to the non-deterministic behaviour of the Java garbage collector. Future benchmarks could employ the Java Microbenchmark Harness (JMH), which supports JVM warm-up, repeated measurements in separate JVM instances, and garbage collection profiling to estimate memory allocations. Additionally, JVM profiling tools and OS-level process monitoring could provide more detailed insights into heap usage and overall process memory consumption, respectively.

The measurements of execution time and memory consumption for Q-learning refer exclusively to the solve phase, i.e. the training performed beforehand (5000 episodes for each instance) is not included in them.

Classical search algorithms systematically explore the state space through a successor function, whereas Q-learning learns action values through repeated interaction with the environment. In the evaluated setting, the additional cost of Q-learning appears primarily during training, while policy execution after training is considerably faster than explicit search.

## 5. Conclusion

The results show that BFS and A* always find the shortest paths, DFS uses less memory, while Q-learning can learn a good policy but requires sufficient training. Q-learning demonstrated that it is possible to learn almost optimal behaviour without any prior knowledge of the environment, but only with a sufficient number of episodes and carefully selected hyperparameters. Perhaps the most important conclusion of this paper is that every approach carries its own trade-offs.

Future work could be directed towards testing dynamic mazes whose structure changes during navigation, and towards the application of deep learning for larger state spaces in which the tabular approach becomes impractical. This paper lays firm foundations for future research, offering a clear picture of where classical search and reinforcement learning meet today and where they still diverge.

## 6. References


[1] Elkari Eidia, B.; Ourabah, L.; Sekkat, H.; Hsane, A.; Essaïouad, C.; Bouargane, Y. & El Moutaouakil, K. (2024). Exploring maze navigation: a comparative study of DFS, BFS, and A* search algorithms, Statistics, Optimization & Information Computing, Vol. 12, No. 3, pp. 761-781, ISSN 2311-004X, DOI: 10.19139/soic-2310-5070-1939

[2] Xu, C. (2023). Maze solving problem using Q-learning, Applied and Computational Engineering, Vol. 6, pp. 1491-1497, ISSN 2755-2721, DOI: 10.54254/2755-2721/6/20230909

[3] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, ISBN 978-0-201-63361-0, Reading, MA, USA

[4] Russell, S. & Norvig, P. (2010). Artificial Intelligence: A Modern Approach, 3rd ed., Pearson, ISBN 978-0-13-604259-4, MA, USA

[5] Stipanicev, D.; Seric, L. & Braovic, M. (2021). Uvod u umjetnu inteligenciju [Introduction to Artificial Intelligence], Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture at the University of Split, Split, Croatia

[6] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. (2009). Introduction to Algorithms, 3rd ed., MIT Press, ISBN 978-0-262-03384-8, Cambridge, MA, USA

[7] Porkolab, I.; Djambic, G. & Kucak, D. (2018). Performance issues and gains of caching the pathfinding data with FAST-N algorithm, In: DAAAM International Scientific Book 2018, Katalinic, B. (Ed.), pp. 309-316, DAAAM International, ISBN 978-3-902734-19-8, Vienna, Austria, DOI: 10.2507/daaam.scibook.2018.27

[8] Hossain, T.; Uddin, S. M. M. & Hasan, A. F. (2021). Comprehensive Review of the A* Algorithm: Theory, Implementation, and Applications, Technical Report, Independent University, Bangladesh, DOI:10.13140/RG.2.2.29143.04003

[9] Hart, P. E.; Nilsson, N. J. & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics, Vol. 4, No. 2, pp. 100-107, ISSN 0536-1567, DOI: 10.1109/TSSC.1968.300136

[10] Watkins, C. J. C. H. & Dayan, P. (1992). Q-Learning, Machine Learning, Vol. 8, pp. 279–292, ISSN 0885-6125, DOI: 10.1007/BF00992698

[11] https://www.mitchellspryn.com/2017/10/28/Solving-A-Maze-With-Q-Learning.html, (2017). Solving A Maze With Q Learning, Accessed on: 2026-06-01


Working Paper of 37th DAAAM Symposium