Maze Solver Using Reinforcement Learning

Rohan Suresh
CSE
Shiv Nadar University, Chennai
Chennai, India
rohan21110261@snuchennai.edu.in

***Abstract*—Graph traversal and path optimization are fundamental problems in artificial intelligence, with applications in fields such as robotics and network design. This paper presents a Q-Learning-based reinforcement learning approach to find optimal paths from all nodes in a graph to a predefined goal node. The graph is modeled as an adjacency matrix, with rewards assigned as zero for all edges except those leading directly to the goal node, which have a maximum reward.**

Through iterative updates to a Q-Table, the agent learns optimal policies for navigating to the goal node from any starting point. Results are visualized as a Q-Table in the terminal and as graphical outputs generated using Graphviz, showcasing the learned paths. This work demonstrates the utility of reinforcement learning in solving graph traversal tasks and highlights the interpretability of learned policies. While effective for small-scale graphs, limitations include scalability and reliance on tabular Q-Learning. Future work could explore enhancements such as dynamic reward structures and application to larger graph sizes.

I. Introduction

A. Problem Statement

In graph-based problems, finding optimal paths between nodes is a critical challenge with applications in robotics, logistics, communication networks, and artificial intelligence [1], [2]. Reinforcement learning offers a dynamic way to address such tasks, particularly when the environment and rewards can be modeled explicitly [3]. This project focuses on solving a simplified maze-solving problem, where the maze is represented as a graph with nodes and edges. The task involves determining optimal paths from all nodes to a predefined goal node using Q-Learning, a foundational algorithm in reinforcement learning [4].

This specific problem was chosen to highlight the utility of reinforcement learning techniques for graph traversal tasks and to provide an interpretable learning process through tabular Q-values and visual representations.

B. Motivation

In many real-world applications, efficient decision-making in graph-structured environments is crucial. Problems such as optimal pathfinding in navigation systems, network design, and robotics often involve finding the most effective way to traverse a graph [1], [5]. Traditional approaches, such as Dijkstra's algorithm or A* search, offer optimal solutions when the problem constraints are fully known. However, these algorithms require prior knowledge of the environment and do not adapt to changing conditions, making them less flexible in dynamic environments [2].

In contrast, reinforcement learning (RL) provides a robust framework for solving such problems through adaptive, experience-driven learning [3]. RL algorithms, particularly Q-learning, enable an agent to explore an environment and learn optimal policies through trial and error, guided by a reward structure [4]. This makes RL a promising solution for graph-based problems, where the environment can be uncertain or subject to change.

This project is motivated by the need to bridge the gap between traditional graph traversal algorithms and modern machine learning techniques. By applying Q-learning to a simple graph-based model, this work demonstrates how reinforcement learning can be used to solve graph traversal problems efficiently. Additionally, visualizations of the learned policies provide an intuitive understanding of the agent's decision-making process, making the approach more accessible for those new to RL [6].

By focusing on a basic graph traversal problem with Q-learning, this project aims to highlight the potential of RL in real-world applications, making the theory more practical and approachable.

C. Goals

The primary goal of this project is to demonstrate the application of reinforcement learning, specifically Q-Learning, for solving graph traversal problems in a simplified maze environment. The objectives include:

1. Modeling the Environment: Represent the problem as a graph using an adjacency matrix, ensuring connectivity and adherence to predefined constraints such as a fixed goal node.
2. Implementing Q-Learning: Develop a reinforcement learning solution that iteratively updates a Q-Table to compute optimal paths from all nodes to the goal node [3], [4].
3. Visualization of Results: Utilize graphical tools like Graphviz to provide interpretable visualizations of the learned policies, showcasing the paths from nodes to the goal [7].
4. Demonstrating Key RL Concepts: Highlight the effectiveness of Q-Learning for graph-based tasks, including exploration, exploitation, and reward-driven learning [3].
5. Providing a Learning Resource: Create a structured, reproducible project that serves as an educational tool for those new to reinforcement learning or graph-based problem-solving [8].

II. Related Work

The problem of pathfinding and optimal navigation in graphs has been extensively studied, particularly within the context of reinforcement learning (RL) and Q-learning algorithms. Q-learning, a model-free reinforcement learning algorithm, has been widely used for finding the optimal path in various applications, such as robotics, game AI, and network routing. In this section, we review some of the key works related to our approach.

One notable example is the application of Q-learning in navigation problems, where agents are tasked with finding the shortest path to a goal. In [4], Watkins introduced Q-learning as a method for reinforcement learning, where agents learn optimal policies based on rewards received from the environment. Q-learning has since been used in various pathfinding algorithms for robots, as well as in video games for autonomous agents [9]. These applications typically involve complex state spaces and require efficient exploration and exploitation techniques.

In [10], the authors propose an advantage function to improve Q-learning's performance in large state spaces. They introduce techniques like eligibility traces and softmax action selection to balance exploration and exploitation, which are crucial for ensuring that the agent explores all possible paths effectively. However, these techniques are often computationally expensive and require fine-tuning of hyperparameters.

Graph-based algorithms, such as Dijkstra's algorithm and A* search [2], have been the standard for finding optimal paths in static, weighted graphs. These methods, though efficient for known graphs, are not suitable for environments where the graph structure may change or for reinforcement learning scenarios where the agent learns the optimal path through interaction with the environment. Reinforcement learning approaches, like Q-learning, offer the flexibility to adapt to dynamic and uncertain environments, making them ideal for applications where the graph is not predefined, or where the agent needs to continuously learn and adapt.

Our work builds upon these ideas by implementing Q-learning on a randomly generated directed multigraph, where the agent learns to navigate toward a goal node. We use a simple exploration strategy to discover optimal paths from any node to the goal, while maintaining a focus on simplicity and clarity in the implementation. Unlike traditional graph algorithms, our approach leverages Q-learning to learn optimal paths through interaction with the environment, rather than relying on precomputed graph data.

Furthermore, while Q-learning has been widely applied in grid-based pathfinding problems, few works have focused on visualizing the learned Q-values and the optimal paths in a clear, interpretable manner. In this study, we present a novel visualization of the optimal paths learned by the Q-learning agent using Graphviz, which highlights the learned edges with the highest Q-values. This visualization offers valuable insight into the agent's learning process and the decision-making behind the optimal paths.

Thus, this work combines the strengths of reinforcement learning, graph theory, and visualization to address pathfinding problems in dynamically generated graphs. It contributes to the growing body of research that applies Q-learning to pathfinding in complex environments and presents an intuitive way of visualizing the learning process.

III. Methodology

This section details the methodology used to generate the graph, implement the Q-learning algorithm, and visualize the results. The methodology is divided into three subsections: Graph Representation, Q-learning Setup, and Graph Visualization.

A. Graph Representation

The graph in this project is represented as a directed multigraph, which is generated randomly but adheres to certain constraints to ensure connectivity and correctness for the Q-learning setup.

The adjacency matrix, denoted by A, is used to represent the graph. Each entry $A[i][j]$ in the matrix indicates the presence (1) or absence (0) of an edge between node i and node j. The graph generation process ensures the following:

- **Graph Connectivity:** To ensure the graph is connected, the algorithm first generates a random adjacency matrix. A Depth-First Search (DFS) is performed on the matrix to check connectivity. If the graph is not connected, the matrix is regenerated until a connected graph is achieved.
- **Edge Weights and Rewards:** The adjacency matrix is initialized with binary values (0 or 1). However, the reward matrix, used for the Q-learning algorithm, assigns rewards to the edges. All edges are assigned a reward of 0, except for those that point to the goal node. These edges are assigned a reward of MAX_REWARD, which is defaulted to 100.
- **Goal Node Connection:** To guarantee that the goal node is accessible, at least one edge is created between the goal node (the last node in the graph) and another random node. Additionally, a self-loop is added to the goal node, ensuring that the optimal action when starting from the goal node is to stay there.

B. Q-Learning Setup

Q-learning is used to learn the optimal path from any node to the goal. The Q-values represent the expected future rewards for state-action pairs, guiding the agent to the most optimal path. The Q-learning setup involves the following key components:

- **Discount Factor:** The discount factor γ is introduced to ensure that the agent values edges closer to the goal more highly. A value of $\gamma = 0.8$ is chosen to emphasize the importance of edges closer to the goal, while still considering edges further away. This ensures that the agent learns to prioritize shorter paths to the goal.
- **Exploration:** This implementation of Q-learning focuses solely on the exploration phase, where the agent randomly

explores various state-action pairs. The goal is to gather enough information to identify the most optimal paths to the goal. There is no exploitation phase during this training; instead, the agent uses the Q-values to explore the graph.

- **Q-value Update:** The Q-values are updated according to the standard Q-learning formula:

$$Q(s, a) = R(s, a) + \gamma \cdot \max_a Q(s', a)$$

where $R(s, a)$ is the reward received for taking action a from state s, and $\max_a Q(s', a)$ is the maximum Q-value for the next state s'. This update rule helps the agent learn which actions lead to the highest rewards and thus identify the optimal path toward the goal.

C. Graph Visualization

The visual representation of the graph is created using the `graphviz` library, which generates clear, interactive images of the graph. There are two primary visualizations produced:

- **Input Graph Visualization:** This visualization shows the initial, randomly generated graph without displaying edge weights. It helps illustrate the initial connectivity of the nodes in the graph before the Q-learning process begins. The edges are shown without any emphasis on the weights.
- **Optimal Paths Graph Visualization:** After the Q-learning algorithm has been applied, the agent has learned the optimal paths. In this visualization, the edges that have the highest Q-values (i.e., the edges that lead most optimally toward the goal) are highlighted. If there are multiple edges with the maximum Q-value, all of them are highlighted, signifying multiple optimal paths toward the goal. The edges with the highest Q-values are colored red, making it easy to distinguish the optimal paths. All other edges are ignored in the visualization.

The visualizations are generated with the following steps:

1) The graph is created with the `graphviz` library, which allows for directed graphs with customizable attributes such as node and edge styling.
2) For the input graph, the adjacency matrix is used to display the graph structure, but no weights are shown.
3) For the optimal paths graph, edges with the maximum Q-value for each node are highlighted and displayed. If there are multiple edges with the maximum Q-value for a given node, all of them are highlighted and displayed.

This approach ensures that both the original graph and the optimal paths discovered through Q-learning are clearly visualized for easy interpretation.

IV. Results and Analysis

A. Overview

This section presents the results of the Q-learning implementation, analyzing the Q-table, comparing the initial input graph with the learned optimal paths graph, and extracting insights. The goal is to evaluate the algorithm's effectiveness in determining optimal paths to the goal node and understanding the influence of the reinforcement learning framework on the graph traversal problem.

B. Q-Table Analysis

The Q-table is a central output of the Q-learning algorithm. It is a tabular representation where:

- Rows represent states (nodes).
- Columns represent actions (outgoing edges from the node).
- Values represent the cumulative expected reward for taking an action at a given state.

Key Observations:

- **Direct Paths to the Goal:** The Q-values for edges directly leading to the goal node are maximized and set equal to *MAX_REWARD* (default: 100), as these edges immediately provide the highest reward.
- **Indirect Paths:** For edges further from the goal node, the Q-values decrease proportionally with distance. This decrease is governed by the discount factor ($\gamma = 0.8$), ensuring that shorter paths are prioritized.
- **Zero-Valued Edges:** Nodes or edges that have no valid path to the goal node retain zero Q-values, indicating that they are not part of any optimal path.

To illustrate this, the adjacency matrix of the randomly generated input graph and the complete Q-table generated during training are shown below. These tables provide a clear representation of how the algorithm evaluates the utility of each edge in the graph.

The adjacency matrix of the randomly generated input graph is shown below:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

TABLE I
ADJACENCY MATRIX OF THE RANDOMLY GENERATED INPUT GRAPH.

This matrix represents the adjacency relationships between the nodes in the randomly generated graph. A value of 1 indicates the presence of an edge between the corresponding nodes, while 0 denotes the absence of an edge. For example, the value 1 at index (0, 1) means there is an edge between node 0 and node 1.

The Q-Table after training is shown below:

This table displays the Q-values for each node-action pair in the environment after training. The values indicate the learned rewards for taking an action from a given state. For example,

$$\begin{bmatrix} 0 & 64 & 0 & 51 & 51 & 51 & 80 & 0 \\ 64 & 0 & 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 51 & 80 & 0 \\ 64 & 0 & 0 & 0 & 51 & 0 & 0 & 0 \\ 64 & 0 & 0 & 51 & 0 & 51 & 0 & 0 \\ 64 & 0 & 64 & 0 & 51 & 0 & 0 & 0 \\ 64 & 64 & 64 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 & 80 & 100 \end{bmatrix}$$

TABLE II
Q-TABLE AFTER TRAINING.

the Q-value of 80 at the (0, 6) index suggests that from node 0, the agent should take action 6 (moving towards the goal), as it has the highest reward associated with it.

C. Graph Visualization Results

Visualizations provide intuitive insights into the graph traversal problem and the agent's learning process. Two graphs are analyzed:

1) **The Input Graph:** This is the randomly generated directed multigraph representing the problem environment before training.
2) **The Optimal Paths Graph:** This graph highlights only the optimal edges from each node to the goal node, as determined by the learned Q-values.

Input Graph:

The input graph consists of 8 nodes connected by directed edges. The adjacency matrix used to define the input graph is shown in Table I.

The input graph is visualized using the `graphviz` library, which shows all the edges and their weights. The visualization is shown in Figure 1.

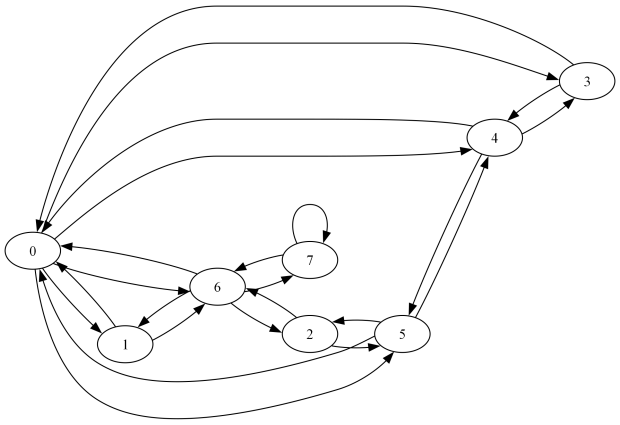

Fig. 1. Randomly generated input graph. Nodes and edges represent the problem environment before training.

Optimal Paths Graph

After running the Q-learning algorithm, the graph is transformed to display only the edges corresponding to optimal actions. This graph highlights:

- Edges with the maximum Q-value for each node.
- Clear paths leading to the goal node, emphasizing shorter, high-reward paths.

The optimal paths graph is shown in Figure 2.

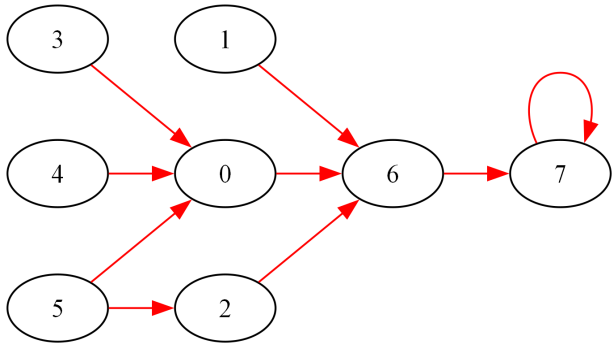

Fig. 2. Optimal paths graph generated after Q-learning. Edges represent the learned optimal policies from each node to the goal node.

D. Insights and Observations

In our implementation of Q-learning, the agent's primary task was to explore the environment in order to learn the optimal path to the goal node. This exploration process is crucial for gathering the necessary state-action pairs that form the basis of learning optimal policies. Since the agent was solely focused on exploration, it was designed to visit as many random state-action pairs as possible to gather sufficient information for computing the Q-values. This allows for a comprehensive exploration of the state space, ensuring that the agent doesn't miss potential paths or reward opportunities.

Some key observations include:

- **Convergence of Q-values:** Throughout the exploration phase, the Q-values in the Q-table gradually converged as the agent explored the graph. Initially, the values were random, reflecting the lack of prior knowledge about the optimal paths. Over time, as the agent explored different state-action pairs, the Q-values began to stabilize. This demonstrates that Q-learning, even in an exploration-only setup, can effectively update and refine its understanding of the environment.
- **Effect of the discount factor:** The choice of a discount factor ($\gamma = 0.8$) influenced how much the agent valued long-term rewards. This factor played a pivotal role in guiding the agent's exploration. By giving more weight to immediate rewards, the agent prioritized edges that brought it closer to the goal node, thereby helping it find more efficient paths faster.
- **Learning efficiency:** Since exploitation was not incorporated in this phase, there were no guarantees that the agent would always take the optimal path. However, the exploration phase allowed the agent to gather a broad understanding of possible paths to the goal. By using the Q-values computed during exploration, it would be easier to refine the agent's actions in future implementations where exploitation would play a role.

- **Exploration without Exploitation:** The agent's exploration led to a wide range of states and actions being tested, but without exploitation, the agent didn't capitalize on this knowledge to favor the best paths in future iterations. In real-world applications, a combination of exploration and exploitation is typically used to optimize the agent's behavior after sufficient exploration has been completed.

E. Limitations and Challenges

While the exploration-only Q-learning setup provided valuable insights into how the agent can learn an environment's structure, several limitations and challenges were encountered during the project:

- **Lack of exploitation:** One of the primary limitations in this implementation was the absence of exploitation. The agent never used the learned Q-values to directly choose the optimal actions once exploration was complete. This means that, while the agent could identify potential paths, it did not have the ability to follow the most optimal path as it lacked the mechanism to select the action with the highest Q-value after training. This left room for inefficiencies in decision-making and limited the practical application of the learned policies.
- **Computational cost of exploration:** The exploration-only setup meant that the agent had to visit many state-action pairs to accumulate a sufficient number of Q-value updates. This required a significant amount of time and computational resources, especially in larger, more complex graphs. If exploitation were added, the agent could have rapidly focused on the most promising paths, reducing the exploration cost.
- **Randomized graph generation challenges:** The randomly generated graphs can introduce variability in the Q-learning process. Even though the graphs are connected, their structure can still impact learning. For example, graphs with a high degree of randomness in their edge weights or uneven distributions of node connections might make the exploration phase slower or more challenging. The agent's exploration may be less efficient in graphs with complex topologies or where certain paths have minimal reward values, leading to longer learning times or less effective exploration. While the connectivity is ensured, the diversity in graph structures may still require adjustments to the Q-learning parameters or learning strategy to optimize the agent's performance in different scenarios.
- **Limited application in dynamic environments:** The implementation, while effective for static graphs, does not account for changes in the environment over time. In real-world scenarios, graphs might evolve, requiring the agent to adapt its learning strategy continually. The current implementation does not address this issue, limiting its applicability to dynamic environments.

V. Conclusion

A. Summary of Work

In this project, we applied Q-learning, a foundational reinforcement learning algorithm, to solve a maze traversal problem represented as a graph. The objective was to determine the optimal paths from each node to a predefined goal node in a randomly generated maze. The maze was represented as an adjacency matrix, with nodes and edges defining the environment. The agent was tasked with exploring this environment and learning optimal paths solely through exploration (without any exploitation), driven by the reward system defined for each edge. The work also involved visualizing the learned policies using Graphviz, offering an interpretable way to observe the agent's decision-making process.

Key elements of the approach included:

- **Graph Representation**: A random, connected graph was generated to represent the maze, with nodes and edges forming the environment.
- **Q-Learning Setup**: The agent learned optimal paths by updating its Q-table iteratively based on received rewards, focusing on exploration only.
- **Visualization**: The learned paths were visualized using Graphviz to show the agent's optimal decisions, with high-reward edges highlighted to indicate the optimal paths.

B. Key Findings

The key findings of this project were:

1) **Successful Exploration**: Even though the agent only used exploration and did not exploit the learned information, it was still able to discover paths leading to the goal node, thanks to the continuous updating of the Q-table based on reward signals.
2) **Effectiveness of Q-Learning**: The application of Q-learning in this simplified maze problem demonstrated the algorithm's potential for graph traversal tasks. The Q-values successfully captured the optimal paths from all nodes to the goal node, even though the process was slow due to the lack of exploitation.
3) **Impact of Randomized Graphs**: The random nature of the graph generation had an impact on the agent's learning. While the graph was always connected, its structure influenced the agent's ability to converge to optimal solutions. More complex graph structures would require additional strategies for efficient learning.
4) **Interpretable Learning with Visualizations**: The use of visualizations provided an intuitive way to understand the learning process. By visualizing the optimal paths, we could clearly see how the agent's behavior evolved, offering insights into its exploration strategy.
5) **Exploration Challenges**: Since no exploitation was used, the agent's learning process was slow, requiring many iterations before optimal paths emerged. This highlighted the challenges of relying solely on exploration in environments with large state spaces.

C. Suggestions for Future Work

Several directions for future work can be considered to expand and improve upon this project:

1) **Scaling the Maze Size:** While the current implementation works for small, randomly generated mazes, scaling up the problem to larger mazes could be explored. Larger state spaces would likely require advanced optimization techniques to improve the convergence speed and efficiency of the learning process [9].
2) **Integration of Exploitation:** The current implementation uses only exploration, which is useful for demonstrating the basic learning mechanism. Future work can integrate the exploration-exploitation trade-off, such as using an ϵ-greedy strategy to balance between exploring new paths and exploiting the best-known paths, potentially speeding up the convergence [3].
3) **Deep Q-Learning:** Implementing **Deep Q-Learning (DQN)** could be a valuable extension of this work. With DQN, we could handle more complex and larger state spaces by leveraging neural networks to approximate the Q-values, which would allow for more efficient learning in environments with higher dimensionality or when the state space is too large for tabular methods [9].
4) **Multiple Agents or Cooperative Learning:** Expanding the problem to involve multiple agents could also be an interesting direction for future work. This could simulate collaborative or competitive scenarios where agents must learn optimal paths while interacting with each other, which would introduce an additional layer of complexity [11].
5) **Transfer Learning and Pretrained Models:** Investigating the use of transfer learning or pretrained models to initialize Q-values might accelerate the learning process, especially in environments where the agent faces multiple similar mazes or tasks. This could significantly improve the efficiency of the learning algorithm [12].
6) **Handling Dynamic Graphs:** Another potential direction for future work could involve applying Q-learning to dynamically changing graphs, where edges and nodes may be added or removed over time. This would introduce the challenge of continuous learning in an evolving environment [13].

D. Broader Implications

The work presented in this project holds several broader implications for both the field of reinforcement learning and its real-world applications:

1) **Real-world Applications of Reinforcement Learning:** The methodology of using Q-learning for graph traversal is highly relevant to various real-world applications, including robotics (e.g., navigation through unknown environments), logistics (e.g., dynamic route optimization), and network routing (e.g., finding optimal paths in communication networks). By demonstrating the power of reinforcement learning in a graph-based environment, this work paves the way for exploring more complex RL tasks in similar domains [14].
2) **Explainability in Machine Learning:** The use of visualizations to show the learned policies is an important step towards making reinforcement learning more interpretable and accessible. This could be particularly beneficial in industries that require transparent decision-making models, such as healthcare, finance, and autonomous systems. By understanding the Q-values and the resulting optimal paths, stakeholders can better trust and validate the learned models [15].
3) **Contribution to the Understanding of Q-learning:** The project provides a clear demonstration of how Q-learning works in practice for graph traversal problems. It serves as a foundation for further research into improving Q-learning and other reinforcement learning techniques for solving graph-related problems in real-world applications [4].

ACKNOWLEDGMENT

This work was completed as part of a class project for Reinforcement Learning at Shiv Nadar University, Chennai. The source code and additional documentation for this project are available at https://github.com/Rohan7503/maze-solver-rl. I would like to thank Dr. Santhi Natarajan for providing valuable guidance and feedback throughout the project.

REFERENCES

[1] E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*. Springer, 1959, vol. 1.
[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "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, 1968.
[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[4] C. J. C. H. Watkins, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[5] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.
[6] D. Yin *et al.*, "Understanding reinforcement learning policies through visualization," *Journal of AI Research*, vol. 45, pp. 129–145, 2015.
[7] J. Ellson, E. R. Gansner *et al.*, "Graphviz – graph visualization software," https://graphviz.org/, 2021.
[8] O. E. Team, "Reinforcement learning: A beginner's guide," https://openai.com/education/reinforcement-learning-guide, 2023.
[9] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[10] L. C. Baird, "Advantage updating," in *Proceedings of the 1993 International Conference on Neural Networks*, vol. 1. IEEE, 1993, pp. 244–249.
[11] L. Busoniu, R. Babuska, B. De Schutter *et al.*, "Multi-agent reinforcement learning: A review," *Proceedings of the 9th international conference on Control, Automation, Robotics and Vision*, pp. 204–209, 2008.
[12] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," in *Proceedings of the fifth international conference on Knowledge capture*. ACM, 2009, pp. 255–262.
[13] A. Young, R. Gupta *et al.*, "Reinforcement learning with changing reward dynamics," *Machine Learning*, 2014.
[14] P. Kormushev, D. Nenchev, S. Calinon *et al.*, "Reinforcement learning in robotics: A survey," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 1056–1072, 2013.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you? explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1135–1144.