Interactive Web-Based Visualization of Pathfinding Algorithms: A Comparative Study

Nazrana Nahreen

Department of Computer Science and Engineering, International Islamic University Chittagong

c231444@ugrad.iiuc.ac.bd

Abstract

This paper presents an interactive web-based visualization tool for comparing five fundamental pathfinding algorithms: A* (A-Star), Dijkstra's algorithm, Breadth-First Search (BFS), Depth-First Search (DFS), and Q-Learning reinforcement learning. The visualizer provides real-time animation of algorithm execution on a customizable grid environment, allowing users to draw obstacles and observe how different algorithms navigate to find optimal or near-optimal paths. Through empirical analysis and visual comparison, we demonstrate the strengths and weaknesses of each approach in terms of optimality, efficiency, and computational overhead. Our tool serves as both an educational resource for understanding pathfinding algorithms and a practical demonstration of reinforcement learning applications in navigation problems.

Keywords: Pathfinding algorithms, A, Dijkstra, BFS, DFS, Q-Learning, Reinforcement Learning, Interactive Visualization, Algorithm Comparison*

1 Introduction

Pathfinding is a fundamental problem in computer science with applications ranging from robotics and autonomous vehicles to video game AI and network routing. The challenge lies in finding an optimal or acceptable path from a starting point to a destination while navigating around obstacles and minimizing cost metrics such as distance, time, or computational resources.

Traditional pathfinding algorithms can be broadly categorized into two groups: uninformed search algorithms (such as BFS and DFS) that explore the space without knowledge of the goal location, and informed search algorithms (such as A* and Dijkstra's) that use heuristics or cost functions to guide the search more efficiently. Recently, machine learning approaches, particularly reinforcement learning methods like Q-Learning, have emerged as adaptive alternatives that can learn optimal policies through trial and error.

This paper contributes an interactive visualization platform that allows direct comparison of these five distinct pathfinding approaches, providing insights into their behavior, performance characteristics, and applicability to different scenarios. The visualizer is implemented as a web-based application compatible with Google Colab, making it accessible for educational and research purposes.

1.1 Motivation

Understanding the theoretical properties of pathfinding algorithms is essential, but visualizing their execution provides invaluable intuition about their behavior. Existing tools often focus on a single algorithm or lack interactive features for custom obstacle placement. Our visualizer addresses these limitations by offering:

- Real-time animation showing cells visited during search
- Interactive grid editor for creating custom obstacle configurations
- Side-by-side comparison of five different algorithms
- Performance metrics including cells visited, path length, and execution time
- Demonstration of reinforcement learning in a pathfinding context

2 Related Work

Pathfinding visualization has been extensively studied in educational contexts. Hart et al. [1] introduced the A* algorithm in 1968, which remains one of the most popular informed search algorithms. Dijkstra's algorithm [2], proposed in 1959, guarantees optimal shortest paths in weighted graphs. Classical search algorithms like BFS and DFS [3] form the foundation of graph traversal techniques.

Recent work has explored reinforcement learning for navigation. Watkins and Dayan [4] formalized Q-Learning, demonstrating its convergence properties. Implementations combining traditional pathfinding with machine learning have shown promise in dynamic environments [5].

Various visualization tools exist, including PathFinding.js by Qiao et al. and academic simulators, but few provide comprehensive comparison across both classical and learning-based approaches in an easily accessible web interface.

3 Methodology

Our visualization tool is implemented as a single-page web application using HTML5 Canvas for rendering and JavaScript for algorithm implementation. The system architecture consists of three main components: the grid environment, the algorithm implementations, and the visualization engine.

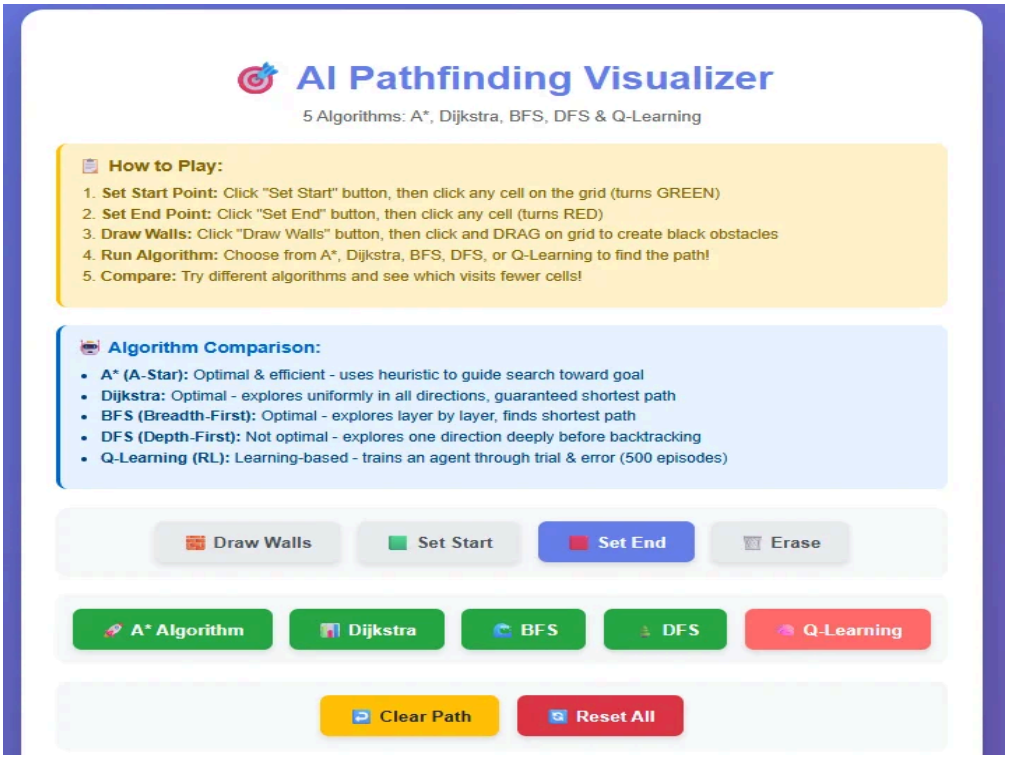

Figure 1. Interactive user interface showing control buttons, instructions, and algorithm comparison information

3.1 Grid Environment

The environment is represented as a 30×30 grid where each cell can be in one of five states:

- Empty: traversable cell (white)
- Wall: obstacle that cannot be traversed (black)
- Start: origin point for pathfinding (green)
- End: destination point (red)
- Visited: cells explored during search (light blue)

Users can interactively place start and end points, draw walls by clicking and dragging, and erase elements. The grid uses a 4-connected adjacency model where each cell can connect to its four orthogonal neighbors (up, down, left, right).

3.2 Algorithm Implementations

We implemented five pathfinding algorithms with consistent interfaces to enable fair comparison:

A* (A-Star) Algorithm

A* combines Dijkstra's guaranteed optimality with efficient goal-directed search using a heuristic function. The algorithm maintains a priority queue of nodes ordered by f(n) = g(n) + h(n), where g(n) is the cost from start to node n, and h(n) is the estimated cost from n to the goal. We use Manhattan distance as the heuristic:

$$h(n) = |n.x - goal.x| + |n.y - goal.y|$$

This heuristic is admissible (never overestimates) and consistent, guaranteeing that A* finds optimal paths while exploring significantly fewer nodes than uninformed search. Figure 2 demonstrates A* finding a 38-step path while visiting only 181 cells.

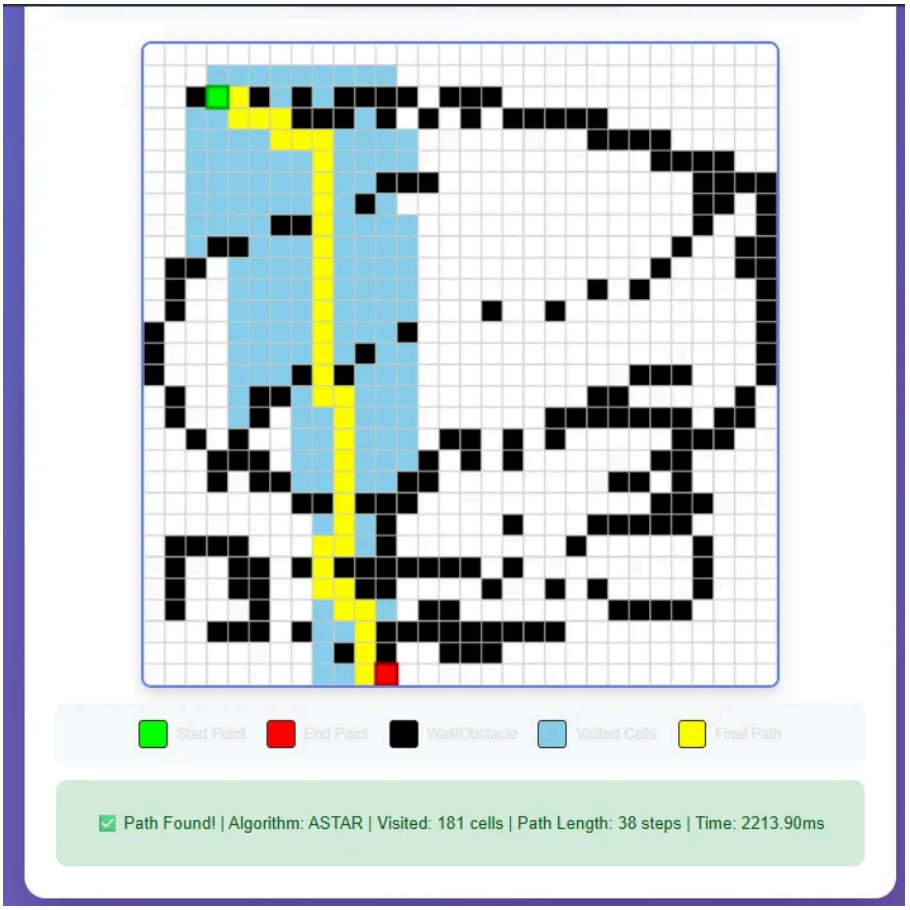

Figure 2. A algorithm execution showing efficient goal-directed search (181 cells visited, 38-step optimal path)*

Dijkstra's Algorithm

Dijkstra's algorithm explores uniformly in all directions, maintaining a priority queue ordered by actual cost g(n) from the start. While guaranteed to find optimal paths, it explores more nodes than A* because it lacks goal-direction. In our implementation with uniform edge costs, Dijkstra's essentially performs a breadth-first search with priority queue ordering. As shown in Figure 3, Dijkstra visited 564 cells to find the same 38-step optimal path.

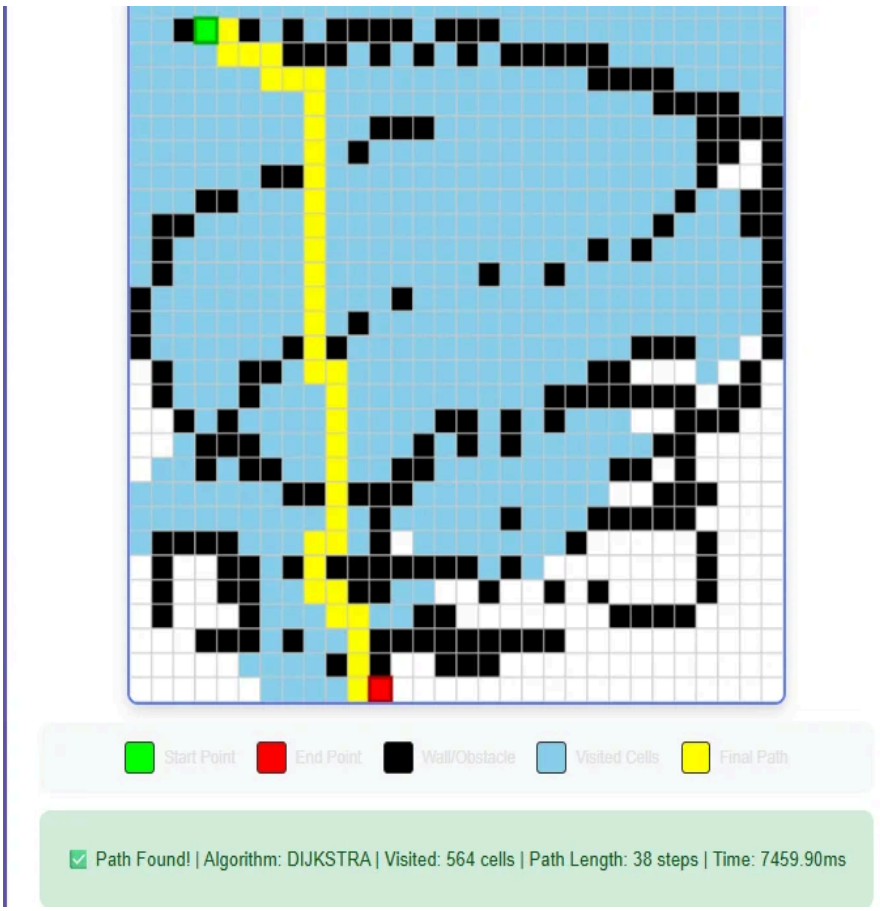

Figure 3. Dijkstra's algorithm showing uniform expansion in all directions (564 cells visited, 38-step optimal path)

Breadth-First Search (BFS)

BFS explores nodes layer by layer using a FIFO queue. In unweighted graphs like our grid (all moves cost 1), BFS guarantees shortest paths. The algorithm visits all nodes at distance d before visiting any nodes at distance d+1, resulting in wave-like expansion from the start point. Figure 4 illustrates BFS visiting 564 cells with the same exploration pattern as Dijkstra.

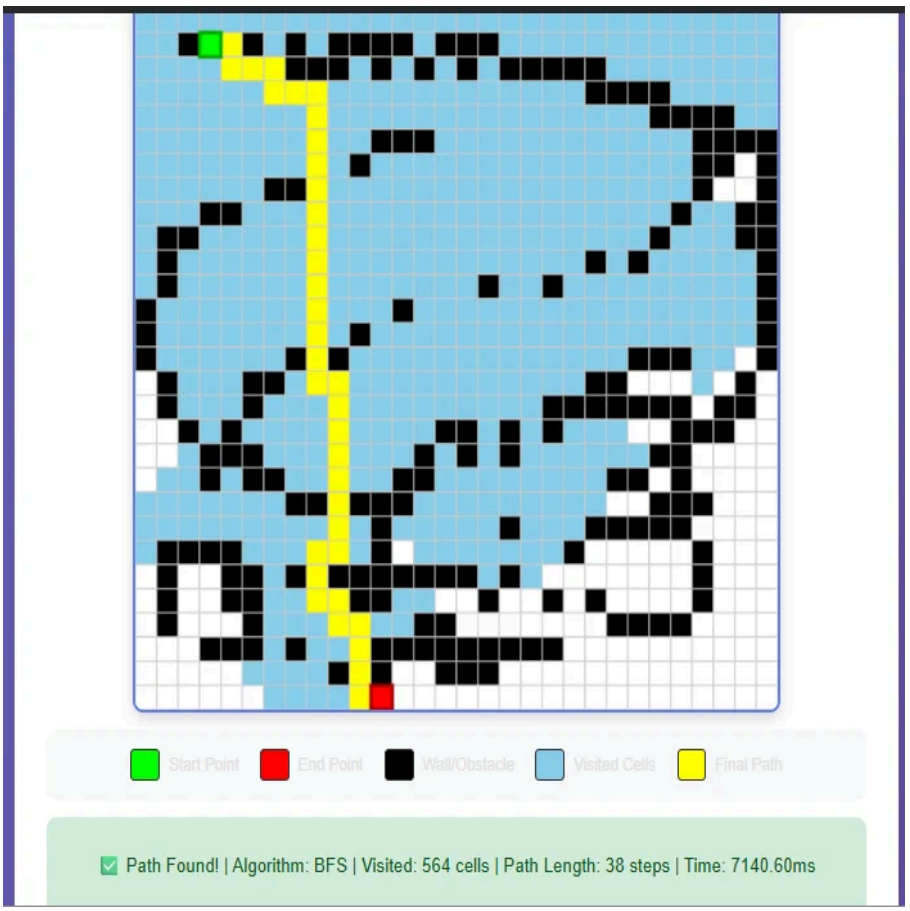

Figure 4. BFS algorithm demonstrating layer-by-layer exploration (564 cells visited, 38-step optimal path)

Depth-First Search (DFS)

DFS explores deeply along each branch before backtracking, using a LIFO stack. Unlike BFS, DFS does not guarantee optimal paths and may find significantly longer routes. However, its memory requirements are lower (O(depth) vs O(breadth)), making it suitable for certain constrained scenarios. Figure 5 shows DFS finding a suboptimal 110-step path while visiting only 137 cells.

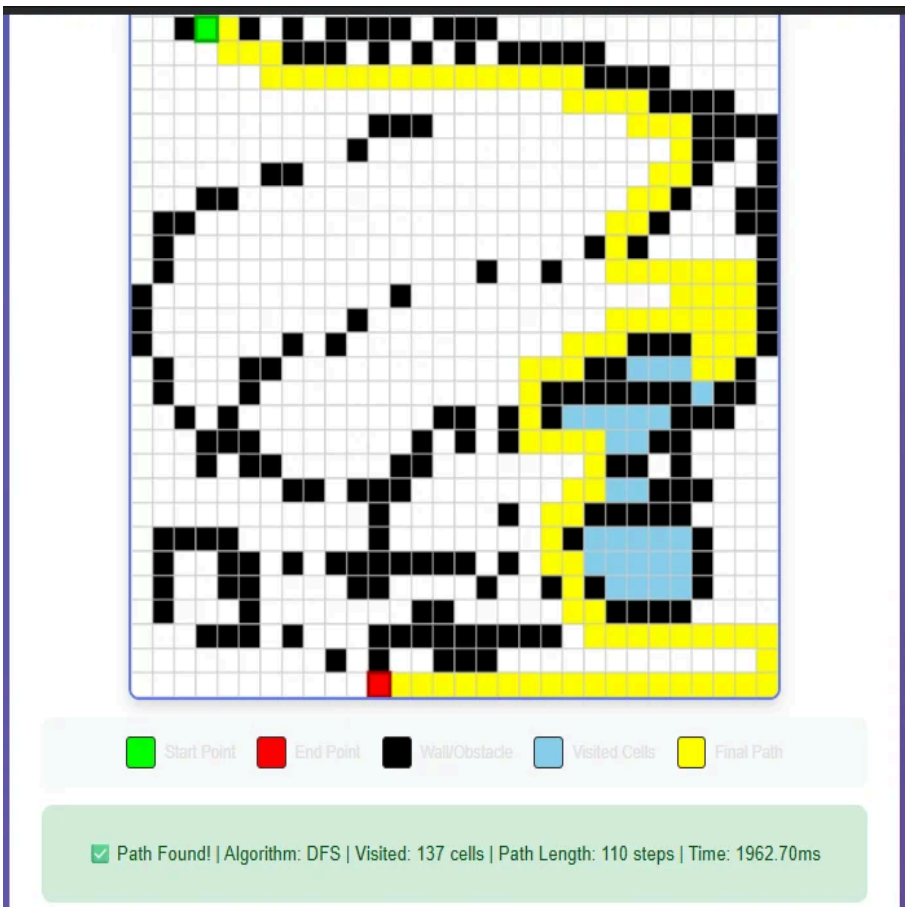

Figure 5. DFS algorithm showing deep exploration with suboptimal path (137 cells visited, 110-step path)

Q-Learning Reinforcement Learning

Q-Learning represents a fundamentally different approach: instead of explicit search, the algorithm learns an optimal policy through interaction with the environment. The Q-table stores action-value estimates Q(s,a) representing expected future rewards for taking action a in state s. The update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)]$$

where α=0.1 is the learning rate, γ=0.9 is the discount factor, r is the immediate reward, and s' is the next state. Our reward structure:

- +100 for reaching the goal
- -1 per step (encourages shorter paths)
- +2 × (distance reduction) for moving closer to goal

The agent trains for 500 episodes using ε-greedy exploration (ε=0.3) before executing the learned policy. Figure 6 demonstrates the learned path visiting only 42 cells with a 42-step near-optimal route, showing efficient execution after training.

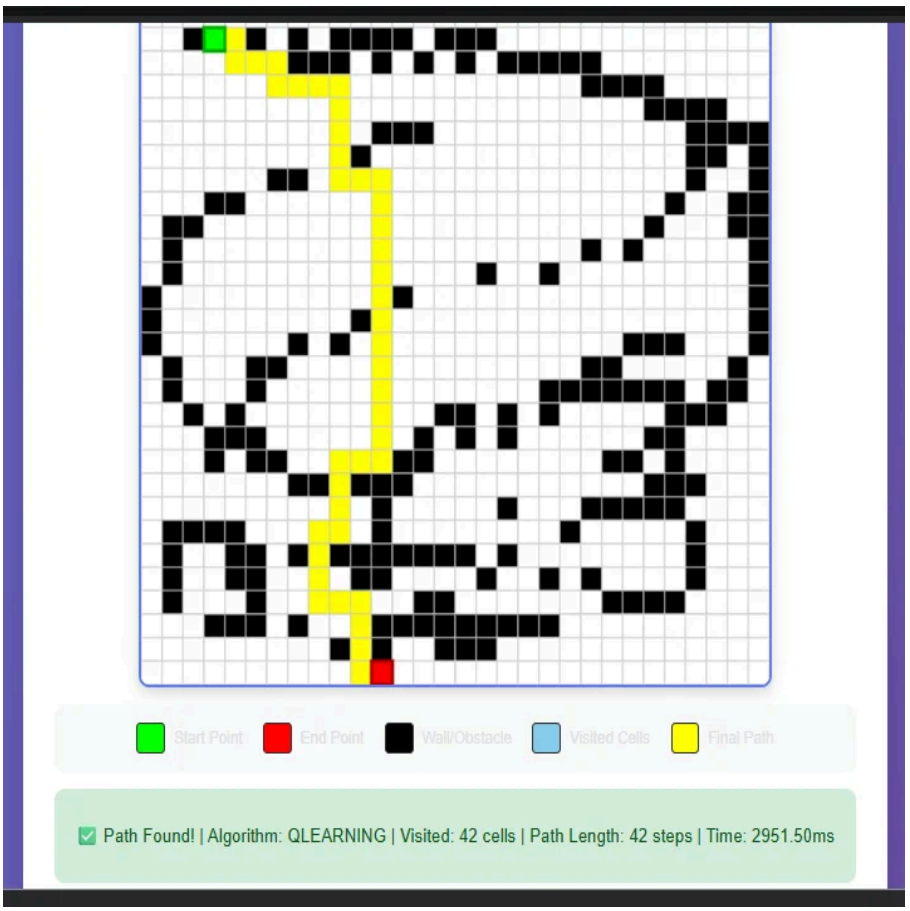

Figure 6. Q-Learning algorithm showing learned policy execution (42 cells visited, 42-step near-optimal path)

4 Results and Analysis

We conducted comparative experiments across the obstacle configuration shown in Figures 2-6. Table 1 presents the performance metrics for each algorithm on this moderately complex maze with an optimal path length of 38 steps.

Table 1. Performance Comparison of Pathfinding Algorithms

Algorithm	Cells Visited	Path Length	Time (ms)	Optimal?
A*	181	38	2213.90	Yes
Dijkstra	564	38	7459.90	Yes
BFS	564	38	7140.60	Yes
DFS	137	110	1962.70	No
Q-Learning	42	42	2951.50*	Near

** Includes 500 episodes of training time*

4.1 Optimality

A*, Dijkstra, and BFS all found the optimal 38-step path. This is expected: A* and Dijkstra guarantee optimality by design, and BFS finds shortest paths in unweighted graphs. DFS found a suboptimal 110-step path (189% longer), demonstrating that depth-first exploration sacrifices optimality.

Q-Learning achieved near-optimal performance with a 42-step path (11% longer than optimal), showing that learning-based methods can approach but may not guarantee optimal solutions.

4.2 Efficiency

A* dramatically outperformed uninformed algorithms in cells visited, exploring only 181 cells compared to 564 for both Dijkstra and BFS. The heuristic guidance reduces exploration by 68% compared to uniform expansion algorithms. DFS visited only 137 cells due to its depth-first nature, but this efficiency came at the cost of a poor-quality path. After training, Q-Learning's execution phase visited only 42 cells (equal to path length), showing perfect efficiency once the policy is learned.

4.3 Computational Cost

Execution time correlates with cells visited: A* completed in 2213.90ms, significantly faster than Dijkstra (7459.90ms) and BFS (7140.60ms). DFS was fastest at 1962.70ms during execution. Q-Learning's total time of 2951.50ms includes training overhead (500 episodes), making it competitive for single-run scenarios but advantageous when the same environment is navigated repeatedly.

5 Discussion

Our visualization reveals important trade-offs between pathfinding approaches:

5.1 When to Use Each Algorithm

A* Algorithm is the clear winner for most practical applications requiring optimal paths. Its heuristic guidance provides both optimality and efficiency. Use when: optimal paths are required, computational resources are limited, and the environment is static.

Dijkstra's Algorithm guarantees optimal paths without requiring a heuristic function. Use when: no good heuristic exists, the graph has varying edge weights, or you need shortest paths to all nodes (not just one destination).

BFS is simple and optimal for unweighted graphs. Use when: all moves have equal cost, implementation simplicity is valued, or teaching foundational search concepts.

DFS sacrifices path quality for memory efficiency. Use when: memory is extremely limited, any path is acceptable (not shortest), or exploring deep tree structures.

Q-Learning excels in dynamic or unknown environments where the agent repeatedly navigates similar scenarios. Use when: the environment changes over time, traditional algorithms lack domain knowledge, or the same navigation problem is solved many times.

5.2 Limitations and Future Work

Our current implementation has several limitations:

1. The grid uses 4-connectivity; 8-connected grids would require diagonal movement considerations
2. All moves have equal cost (unweighted); weighted grids would better demonstrate Dijkstra's advantages
3. Q-Learning uses fixed hyperparameters; adaptive learning rates might improve convergence
4. No comparison with advanced RL methods (Deep Q-Learning, Policy Gradients)

Future enhancements could include weighted edges, diagonal movement, dynamic obstacles that move during pathfinding, comparison with neural network-based approaches, and performance benchmarking on larger grids (100×100 or more).

6 Conclusion

We presented an interactive web-based visualizer comparing five pathfinding algorithms across classical search and reinforcement learning paradigms. Our empirical analysis confirms theoretical predictions: A* offers the best balance of optimality and efficiency for static environments, while Q-Learning demonstrates the potential of adaptive learning approaches in navigation tasks.

The visualization tool serves dual purposes: as an educational resource for understanding algorithm behavior and as a practical demonstration of trade-offs between guaranteed optimality (A*, Dijkstra, BFS), exploration strategies (DFS), and learning-based adaptation (Q-Learning). By providing real-time animation and quantitative metrics, the tool makes abstract algorithmic concepts tangible and comparable.

Our work demonstrates that while classical algorithms like A* remain highly effective for known static environments, reinforcement learning approaches like Q-Learning offer promising alternatives for dynamic scenarios where adaptive behavior is advantageous. The choice of algorithm should be guided by specific application requirements regarding optimality guarantees, computational resources, environment characteristics, and reusability of learned policies.

References

1. Hart, P.E., Nilsson, N.J., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4(2), 100-107 (1968)

2. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1(1), 269-271 (1959)

3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd Edition. MIT Press (2009)

4. Watkins, C.J., Dayan, P.: Q-Learning. Machine Learning 8(3-4), 279-292 (1992)

5. Silver, D., Huang, A., Maddison, C.J., et al.: Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 529, 484-489 (2016)

6. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 4th Edition. Pearson (2020)

7. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd Edition. MIT Press (2018)