Path Planning Algorithms: Analysis and Evaluation

Dhrumil Kotadia

April 15, 2025

1 Introduction

Path planning is a fundamental problem in robotics and autonomous systems, where the objective is to navigate from a start point to a goal while avoiding obstacles. This project evaluates the performance of three path planning algorithms — Breadth-First Search (BFS), Dijkstra, and Rapidly-exploring Random Trees (RRT) — on grid maps representing binary occupancy grids. The robot has a 1-meter diameter and the maps consist of 1m x 1m grid cells.

2 Problem Statement

As outlined in the Module 1 Project description, the task involves: Implementing collision detection for the robot navigating provided grid maps, developing classical (BFS, Dijkstra) and sampling-based (RRT) path planning algorithms, comparing algorithm performance based on computation time, path length, and memory usage and generating visualizations of paths computed by each algorithm.

3 Implementation Overview

The implementation was done in Python using NumPy and Matplotlib. Grid parsing, path computation, and visualizations were performed using modular functions in `path_planner.py` and `utils.py`. The movement model used was 8-connected for all algorithms to enable diagonal transitions.

4 Algorithms Implemented

4.1 Breadth-First Search (BFS)

BFS is a uniform-cost search that explores neighbors level-by-level. It guarantees shortest path (in steps) in an unweighted grid. Results for BFS can be observed in the following figures.

4.2 Dijkstra's Algorithm

Dijkstra extends BFS by accounting for cost to reach each node, ensuring optimal paths in weighted graphs. On uniform-cost grids, its performance is close to BFS.

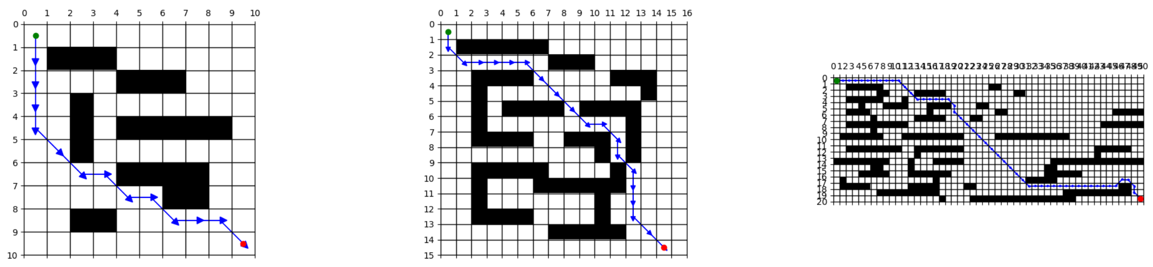

Figure 1: BFS Path Planning Results on Map 1, 2 and 3

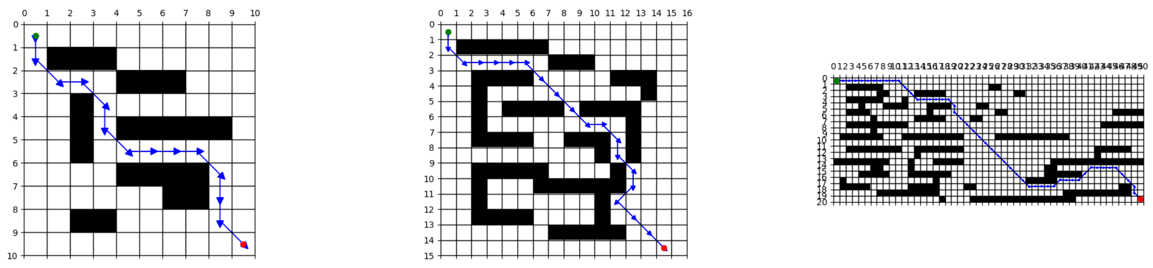

Figure 2: Dijkstra Path Planning Results on Map 1, 2 and 3

4.3 Rapidly-exploring Random Trees (RRT)

RRT is a sampling-based method that builds a tree by exploring random samples. It is well-suited for high-dimensional or continuous spaces, though it does not guarantee optimality.

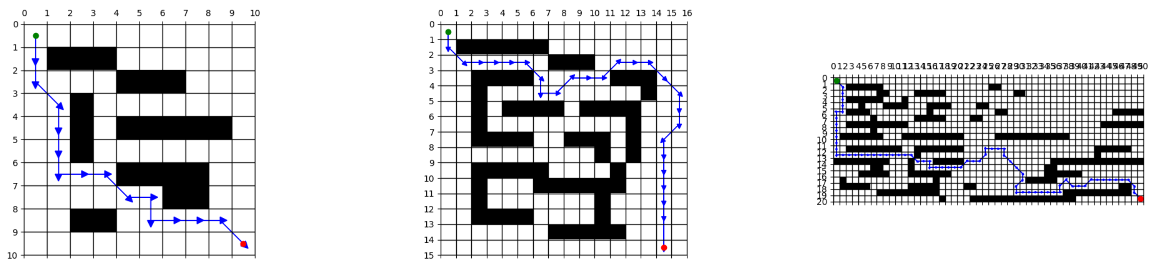

Figure 3: Dijkstra Path Planning Results on Map 1, 2 and 3

5 Experimental Results

Three maps were tested. Each algorithm was evaluated for time taken, path length, and memory usage.

5.1 Computation Time (seconds)

Algorithm	Map1	Map2	Map3
BFS	0.00024	0.000504	0.00232
Dijkstra	0.00031	0.000625	0.00265
RRT	0.00697	0.030426	1.36624

5.2 Path Length (meters)

Algorithm	Map1	Map2	Map3
BFS	13.071	19.485	58.870
Dijkstra	13.071	20.314	60.527
RRT	19.071	20.899	66.142

5.3 Memory Usage (MB)

Algorithm	Map1	Map2	Map3
BFS	0.000128	0.000192	0.033296
Dijkstra	0.000128	0.000192	0.016390
RRT	0.001538	0.001538	0.014786

6 Discussion

6.1 Time Efficiency

BFS and Dijkstra are highly time-efficient for small and medium maps, with execution times in the millisecond range. RRT, by contrast, exhibits significantly higher computation times, especially for larger maps due to its random sampling and collision checking.

6.2 Path Quality

BFS and Dijkstra produced near-optimal paths. BFS generally returns slightly shorter paths due to level-order expansion. RRT paths were consistently longer due to the lack of global optimization and reliance on random sampling.

6.3 Memory Usage

Dijkstra proved most memory-efficient in larger maps due to its prioritized exploration. BFS used more memory due to broader search fronts. RRT, despite being slower, had moderate memory usage due to limited node expansion.

6.4 Invalid Start and Goals

All algorithms were able to detect invalid start and goals in the map. There is an initial check in each planner that ensures that both start and goal states are not invalid. If they are invalid, it automatically returns a "No Path Found" message, ensuring unnecessary computation does not happen. In case that both start and goal states are valid but no path exists from start to goal, all planners will execute until their terminating conditions (eg. RRT will run for max_iterations).

7 Conclusion

This project demonstrated and analyzed classical and sampling-based path planning algorithms. BFS and Dijkstra offer fast and reliable results in grid-based structured maps. RRT is more suitable for unstructured or continuous environments, though at a cost in time and path quality. Future work can explore RRT* for better path optimality and dynamic replanning.

8 Appendix

- Code Files: `path_planner.py`, `utils.py`
- Visual Outputs: Plots for each algorithm are saved as PNGs.
- Map Files: Three occupancy grid maps used for testing (These files are split into grid files and start_goal files which are read by the code).