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.

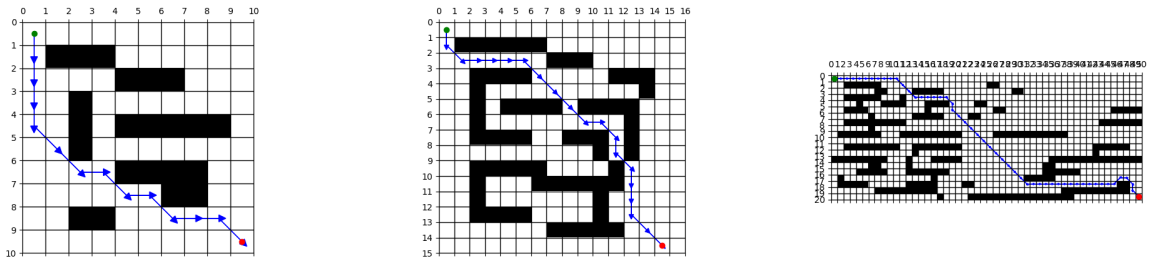

Figure 1: BFS Path Planning Results on Map 1, 2 and 3

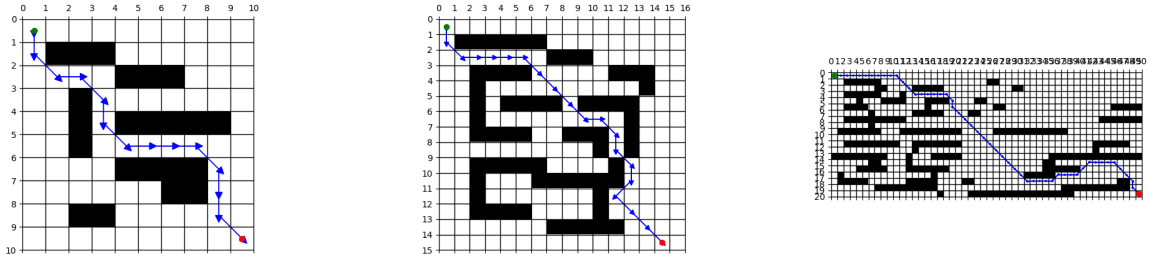

Figure 2: Dijkstra Path Planning Results on Map 1, 2 and 3

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.

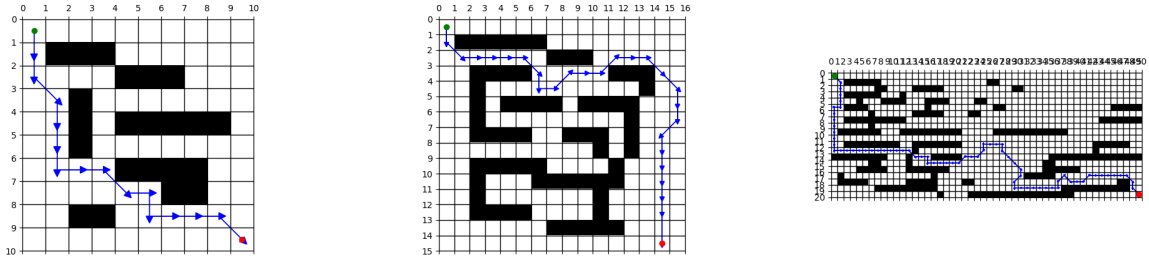

Figure 3: Dijkstra Path Planning Results on Map 1, 2 and 3

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 Analysis

6.1 Time Efficiency

BFS and Dijkstra exhibit high time efficiency, especially in small to medium-sized maps where the number of nodes is limited. BFS uses a queue to explore nodes in a level-order fashion, ensuring uniform expansion, which allows it to find the shortest path quickly in environments where all actions have equal cost. Dijkstra, while similar in approach, uses a priority queue (min-heap) to always expand the node with the least cumulative cost, which can add slight overhead due to heap operations but enables it to handle weighted maps more accurately.

RRT, on the other hand, shows significantly higher computation times. This is primarily due to its reliance on random sampling, nearest-neighbor searches, and frequent collision checks against obstacles. As the size and complexity of the map grow, the number of samples required to find a feasible path increases, leading to longer planning times. Moreover, the lack of deterministic exploration in RRT contributes to variability in planning time across different runs.

6.2 Path Quality

BFS and Dijkstra consistently produce high-quality paths in terms of length and directness. BFS tends to produce slightly shorter paths in uniform-cost maps due to its even expansion in all directions, which can sometimes discover goal-adjacent nodes faster. Dijkstra's paths are optimal in terms of cumulative cost, especially in weighted environments.

RRT paths, however, are noticeably sub-optimal. The algorithm prioritizes exploration over exploitation, leading to jagged, inefficient paths that often require post-processing (e.g., smoothing or shortcutting) to be viable in real-world scenarios. This is expected behavior, as the original RRT algorithm does not incorporate path refinement or cost-based optimization during growth.

6.3 Memory Usage

Dijkstra's use of a priority queue enables targeted exploration and pruning of redundant paths, making it relatively memory-efficient for larger maps. It avoids unnecessary expansion by always selecting the lowest-cost node, reducing the overall number of explored states.

BFS, while fast, maintains a broad search front and stores all frontier nodes in a queue. This leads to increased memory consumption, especially in maps with wide open spaces or multiple equally short paths to the goal.

RRT, although computationally intensive, uses less memory compared to BFS as it only maintains the tree structure of sampled nodes. It does not need to store the full grid or explore exhaustively, which helps contain memory usage. However, as the number of iterations grows, the memory requirements can scale significantly depending on the number of nodes sampled and stored.

6.4 Invalid Start and Goals

Each algorithm begins with robust validation checks to handle invalid start or goal states. If either of these states lies on an obstacle or outside the bounds of the map, the planner immediately returns a "No Path Found" message. This validation step is crucial in preventing unnecessary computation and avoiding misleading results.

When both start and goal positions are valid but a feasible path does not exist due to obstacles or disconnected regions, the behavior of each algorithm diverges slightly:

- **BFS** and **Dijkstra** will exhaustively search the reachable space. If the goal is unreachable, they will terminate once all valid nodes are expanded.
- **RRT** continues to attempt random sampling until the maximum iteration limit is reached. This can lead to wasted computation in hopeless scenarios but is necessary due to the non-deterministic nature of the algorithm.

These safeguards ensure the reliability and robustness of each algorithm when dealing with edge cases and invalid input configurations.

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).