ME5413 Autonomous Mobile Robotics

Homework 3

Name	Matric No
Sun Cheng	A0303599U

A THESIS SUBMITTED FOR AUTONOMOUS MOBILE ROBOTICS

COLLEGE OF DESIGN AND ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

March 23, 2025

Chapter 1

Task1 Global Planning

Implementation Details

In this task, I implemented two path planning algorithms: **A*** and **Dijkstra**. Both algorithms were applied to the given VivoCity level 2 map to compute the shortest paths between five key locations: `start`, `snacks`, `store`, `movie`, and `food`. Each cell in the map represents a 0.2m x 0.2m area, and the path planning algorithms followed an 8-connected grid configuration.

A* Algorithm

Key Features:

- Utilized the Euclidean distance as the heuristic function to guide the search.
- Implemented efficient data structures such as `heapq` for fast priority queue operations.
- The algorithm computed the total distance, visited cells, and runtime for each route.

Improvements:

- Introduced diagonal movement with a cost of 0.282m, improving the path smoothness and reducing travel distance.
- Fine-tuned the heuristic function to balance optimality and computational efficiency.

Dijkstra Algorithm

Key Features:

- Dijkstra's algorithm was implemented by removing the heuristic component from A*, effectively making it a uniform-cost search.
- The algorithm efficiently explored the entire map to guarantee the shortest path result.

Improvements:

- Added dynamic path visualization for improved result interpretation.
- Enhanced cell visit tracking for performance analysis.

Comparison of Algorithms

Metric	A* Algorithm	Dijkstra Algorithm
Total Travel Distance	2815.80 m	2613.60 m
Total Cells Visited	53,720	173,730
Total Runtime	3.82 s	10.62 s

Table 1.1: Algorithm Performance Comparison

Observations:
The A* algorithm outperformed Dijkstra in terms of speed and cell exploration efficiency while still achieving competitive path quality.
Dijkstra's algorithm produced slightly shorter overall distances but required significantly higher computational effort.

Difficulties Encountered and Solutions

Path Deviation in A*: The initial A* implementation occasionally deviated from optimal routes. This was addressed by refining the heuristic function to better estimate remaining costs.

Performance Bottleneck in Dijkstra: Dijkstra's exhaustive exploration slowed the computation considerably. Optimizing data structures and early termination conditions improved performance.

Identified Shortcomings and Suggested Improvements

Shortcomings: The current implementations are limited to static grids and cannot adapt to dynamic obstacles.

Suggested Improvements: Implement dynamic re-planning strategies such as D* Lite for better adaptability in dynamic environments.

Shortest Distance Matrix

	Start	**Snacks**	**Store**	**Movie**	**Food**
Start	0.0	124.6	131.6	157.0	194.4
Snacks	124.6	0.0	99.2	94.4	113.0
Store	131.6	99.2	0.0	192.4	103.2
Movie	157.0	94.4	192.4	0.0	97.0
Food	194.4	113.0	103.2	97.0	0.0

Table 1.2: Shortest Distance Matrix (m)

Chapter 2

Task2 The “Travelling Shopper” Problem

Problem Modeling

The ”Travelling Shopper Problem” can be modeled as a variation of the well-known **Travelling Salesman Problem (TSP)**. The goal is to find the shortest route that starts at the designated point (start), visits each of the given locations (`snacks`, `store`, `movie`, `food`) exactly once, and then returns to the starting point.

Graph Representation

- Each location is treated as a **node** in the graph.
- The shortest distance between each pair of locations (calculated in Task 1) is treated as the **edge weight**.
- The objective is to minimize the total travel distance while visiting all nodes exactly once and returning to the starting point.

Applied Methods

Method 1: Permutation Algorithm (Exhaustive Search)

Implementation Details:
This method enumerates all possible permutations of the locations. For each permutation, the total travel distance is calculated. The permutation with the shortest distance is identified as the optimal route.

Results:

- **Optimal Route:** Start $\rightarrow$ Snacks $\rightarrow$ Movie $\rightarrow$ Food $\rightarrow$ Store $\rightarrow$ Start
- **Total Distance:** 550.80 meters
- **Runtime:** 2.4 seconds

Method 2: Nearest Neighbor Algorithm (Greedy Approach)

Implementation Details:
Starting from the `start` location, the algorithm iteratively selects the closest unvisited location. After visiting all points, it returns to the starting location.
Results:

- **Optimal Route:** Start $\rightarrow$ Snacks $\rightarrow$ Movie $\rightarrow$ Food $\rightarrow$ Store $\rightarrow$ Start
- **Total Distance:** 550.80 meters
- **Runtime:** 2.3 seconds

Comparison of Methods

Aspect	**Permutation Algorithm**	**Nearest Neighbor Algorithm**
Optimality	Guaranteed optimal	May yield suboptimal results
Efficiency	Slow for larger maps	Faster in practical cases
Complexity	$O(n!)$	$O(n^2)$

Table 2.1: Comparison of Methods

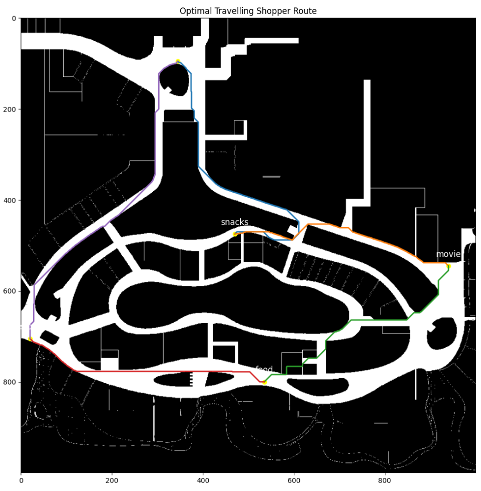
Both methods achieved identical optimal routes in this case, highlighting the effectiveness of the greedy algorithm in this scenario. The nearest neighbor algorithm achieved similar performance but with a slightly shorter runtime, indicating its practicality for problems with moderate complexity.

Final Route Visualization

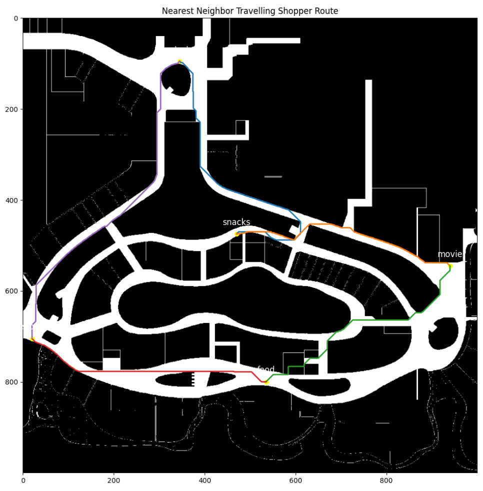
The following visualizations depict the optimal routes obtained:

Conclusion

In this task, both the permutation algorithm and the nearest neighbor algorithm successfully identified the optimal route. While the permutation algorithm offers guaranteed optimality, the nearest neighbor method demonstrated efficient performance with a comparable result. For larger problem scales, the nearest neighbor algorithm is preferable due to its faster runtime.



(a) Permutation Algorithm Route



(b) Nearest Neighbor Algorithm Route

Figure 2.1: Results route for two methods