Hiwi-report

Julen Costa

March 2024

1 Introduction

This report summarises the decisions I have taken during the implementation for the path planning problem.

2 Task Setting 1

The primary objective of Task Setting 1 is to implement a path planning algorithm capable of finding the shortest path between two points within a 2D grid. To accomplish this, I have employed the A* algorithm, a widely-used search algorithm renowned for its efficiency and optimality in pathfinding scenarios.

2.1 A* Algorithm Overview

The A* algorithm is a heuristic-based search algorithm that combines the advantages of both Dijkstra's algorithm and Greedy Best-First Search. It operates by maintaining a priority queue of open nodes and iteratively exploring the node with the lowest total cost, which is the sum of the cost to reach the node from the start node (known as the "g-value") and an estimated cost to reach the goal from the current node (known as the "h-value"). By doing so, A* intelligently navigates towards the goal while considering the shortest path found so far.

2.1.1 Heuristic Function

In the implementation of A*, an essential component is the heuristic function, which estimates the cost of reaching the goal from a given node. For this task, I have utilized the Manhattan distance heuristic, denoted by $h(n)$, calculated as:

$$h(n) = |n_{goal_x} - n_{current_x}| + |n_{goal_y} - n_{current_y}|$$

where n_{goal_x} and n_{goal_y} represent the x and y coordinates of the goal node respectively, and $n_{current_x}$ and $n_{current_y}$ represent the x and y coordinates of the current node being considered. The Manhattan distance heuristic provides a conservative estimate of the remaining distance to the goal by measuring the horizontal and vertical distances between the current node and the goal.

By incorporating the Manhattan distance heuristic into the A* algorithm, the search efficiently explores promising paths while ensuring optimality in finding the shortest path between two points in the 2D grid.

A* algorithm typically outperforms Dijkstra's algorithm in terms of time complexity due to its ability to efficiently explore paths guided by a heuristic function. While both algorithms share the same worst-case time complexity ($\mathcal{O}((V + E)logV)$), A* reduces exploration by using heuristics to guide the search towards the goal, resulting in faster computation times.

3 Task Setting 2

In this task, the objective expands on the previous scenario by introducing obstacles, making it more challenging. Unlike the previous scenario, encoding a heuristic function that satisfies the necessary constraints for optimality (consistency) becomes impractical. This problem is formally known as the Traveling Salesman Problem (TSP), which is NP-hard, implying that there is no known polynomial-time algorithm to solve it optimally.

To tackle this problem, the implemented approach involves first computing pairwise distances between all significant points, including the start and end nodes and the waypoints. Subsequently, well-established algorithms are applied to solve an instance of the TSP problem, aiming to find the shortest path that visits all waypoints and terminates at the destination node.

Two main algorithms have been implemented for this purpose:

Held-Karp Algorithm

The Held-Karp algorithm is an exact algorithm that guarantees optimality. It operates with a computational cost of $\mathcal{O}(n^2 * 2^n)$, where n represents the number of waypoints. Despite its optimality, the Held-Karp algorithm becomes impractical as the number of waypoints increases, typically becoming unmanageable beyond a certain threshold.

Ant Colony Optimization (ACO)

Ant Colony Optimization is a heuristic algorithm inspired by the foraging behavior of ants. It belongs to the category of metaheuristic algorithms, providing solutions that may not always be optimal but are often of high quality. ACO efficiently handles scenarios with a large number of waypoints, making it suitable for real-world applications. However, its trade-off lies in the inability to guarantee optimality in all cases.

In summary, while the Held-Karp algorithm ensures optimality but becomes computationally demanding for large instances, Ant Colony Optimization offers a scalable solution with acceptable performance, albeit without the guarantee of optimality.

4 Task Setting 3

In this final task setting, the objective is to enhance the basic path planning implementation described in Section 2 by incorporating terrain data. Imagine yourself as a leisurely traveler seeking to reach the destination F from your starting point S while minimizing elevation gain. Here, I've introduced six levels of elevation by default, and modified the original A* algorithm to consider elevation gain.

The key question here is: "What is preferable? A path that, although not the shortest, minimizes elevation changes? Or a path that minimizes distance, even if it involves frequent changes in elevation?" To address both objectives simultaneously, I've adapted the A* algorithm using the following heuristic function:

$$heuristic = |n1.terrain\_level - n2.terrain\_level|^k \times (|x_1 - x_2| + |y_1 - y_2|)^{1-k}$$

where:

- $n1.terrain\_level$ and $n2.terrain\_level$ represent the terrain levels of the nodes being considered.
- (x_1, y_1) and (x_2, y_2) are the coordinates of the nodes.
- k is a parameter that allows adjusting the preference between minimizing elevation gain ($k = 1$) and minimizing distance ($k = 0$). Adjusting k enables us to fine-tune the trade-off between distance and elevation.

Setting $k = 1$ prioritizes minimizing elevation gain, while setting $k = 0$ minimizes distance similarly to Task Setting 1, disregarding elevation. The flexibility in choosing k allows for accommodating preferences in the distance-elevation trade-off, making the algorithm adaptable to various scenarios and user preferences.