Project Report: Autonomous Delivery Agent

Student name: Somya

Reg no: 24MIM10226

Slot: B11+B12+B13

Course Instructor: Dr. Rudra Kalyan Nayak

Course:CSA2001 Fundamentals in AI Ml

Introduction

This report details the design and implementation of an autonomous delivery agent capable of navigating a 2D grid environment. The agent's core functionality is built upon several classical search algorithms: Breadth-First Search (BFS), Uniform-Cost Search (UCS), and A* Search with an admissible heuristic. The project also includes a proof-of-concept for dynamic replanning using a local search strategy to handle unforeseen obstacles. Experimental results and a comparative analysis are presented to evaluate the performance of each algorithm across different map instances.

1. Environment Model

The agent operates within a 2D grid environment, model by the Grid class. This class is responsible for loading the environment from a simple text file format. The grid represents a city with distinct cell types and associated movement costs, providing a static but varied environment for the agent.

Each cell on the grid is defined by a coordinate (x,y) and contains one of the following elements:

- **Start (S):** The initial position of the agent.
- **Goal (G):** The destination for the package delivery.
- **Static Obstacle (X):** An impassable cell that the agent cannot enter.
- **Terrain Cost :**A numerical value representing the cost of traversing that cell.

The agent's movement is restricted to four-connected (up, down, left, right) cells. The cost of moving from one cell to an adjacent cell is the integer value of the destination cell. This model allows for the simulation of varying terrains, such as roads (1) or rougher ground (>1), influencing the agent's pathfinding decisions.

2. Agent Design

The agent's decision-making is centered around a Planner class hierarchy. The base Planner class provides common functionality like path reconstruction and result logging. Each specific planning algorithm (BFS, UCS, A*, Local Search) is implemented as a subclass, allowing for easy comparison and modularity.

- **Node Class:** This is a fundamental data structure for the search algorithms. Each Node object represents a specific state in the search space. It stores:
    - The cell coordinates (x,y).
    - The path cost from the start node (g(n)).
    - A reference to its parent node, allowing for the reconstruction of the final path.
- **Agent Class:** This class contains the implementations of the various search algorithms. It takes the Grid as input and uses the Node class to manage its exploration of the state space. It is capable of executing the following strategies:
    - **Uninformed Search (bfs, ucs):** These methods explore the grid without any knowledge of the goal's location.
    - **Informed Search (a_star):** This method uses a heuristic function to guide its search, making it more efficient.
    - **Local Search (hill_climbing_replanning):** This strategy is used for dynamic replanning when an unexpected obstacle appears.

3. Heuristics Used

An admissible heuristic is crucial for the A* search algorithm to guarantee an optimal solution. A heuristic function h(n) is considered admissible if it never overestimates the cost to reach the goal from any node n.

For this project, the **Manhattan distance** is used as the heuristic function for the A* search algorithm. It is defined as:

$h(n)=|x\ n-x\ g|+|y\ n-y\ g|$

where (x n, y n) are the coordinates of the current node and (x g, y g) are the coordinates of the goal node.

The Manhattan distance is admissible because the agent can only move horizontally and vertically (4-connected movement). The shortest possible path from a node to the goal on a grid will always be at least the sum of the horizontal and vertical distances. This heuristic does not overestimate the true cost, thus upholding the optimality guarantee of the A* algorithm. The uniformed search algorithms (BFS, UCS) do not utilize a heuristic.

4. Experimental Results

To evaluate the performance of each algorithm, a series of experiments were conducted on the provided test maps. The key metrics recorded were: Path Cost, Nodes Expanded, and Time taken.

Table 1: Algorithm Performance on small.txt

Algorithm	Path Cost	Nodes Expanded	Time(s)
BFS	9	23	0.0001s
UCS	9	23	0.0001s
A*	9	19	0.0001s

Table 2: Algorithm Performance on medium.txt

Algorithm	Path cost	Nodes Expanded	Time(s)
BFS	39	394	0.0015s
UCS	39	394	0.0019s
A*	39	165	0.0017s

Table 3: Algorithm Performance on large.txt

Algorithm	Path Cost	Nodes Expanded	Time(s)
BFS	99	2500	0.0062s
UCS	99	2500	0.0097s
A*	99	998	0.0050s

Table 4:Algorithm Performance on dynamic.txt

Algorithm	Path Cost	Nodes Expanded	Time(s)
BFS	39	400	0.0022
UCS	39	400	0.0013

A*	39	155	0.0009

5. Analysis and Conclusion

The experimental results highlight the fundamental differences between uninformed and informed search strategies.

- **BFS vs. UCS:** BFS is a classic graph traversal algorithm that finds the shortest path in terms of the number of edges. It expands nodes level by level, making it well-suited for unweighted graphs. However, it does not account for variable terrain costs, which is why its path cost is often suboptimal. UCS, by prioritizing nodes with the lowest cumulative cost, is guaranteed to find the path with the lowest cost, making it optimal for weighted graphs. The trade-off is that UCS may explore more nodes than BFS if the cost values are high, as it doesn't have a heuristic to guide it towards the goal.
- **A* Search:** The A* algorithm, by using the admissible Manhattan distance heuristic, combines the benefits of both BFS and UCS. It intelligently explores the most promising paths first, significantly reducing the number of nodes expanded compared to the uninformed algorithms. As seen in the results, A* consistently finds the optimal path with considerably fewer expanded nodes and in less time, especially on larger maps. The heuristic allows A* to "cut corners" in its search space, leading to superior performance without sacrificing optimality.
- **Dynamic Replanning (Hill-Climbing):** For the dynamic map, a local search strategy like hill-climbing was used. When an unexpected obstacle appears, a new search is initiated from the agent's current position to the goal. While not guaranteed to find the globally optimal path (it can get stuck in local optima), it is a fast and effective method for handling unforeseen changes in the environment. This rapid re-planning capability is essential for a real-world agent that must react quickly to dynamic events, even if it means accepting a slightly suboptimal path.

In conclusion, the choice of algorithm depends on the problem's constraints. For static environments where optimality is paramount, A* is the superior choice. For dynamic, real-time environments, a fast replanning strategy, even if it's suboptimal, is necessary for robust performance. This project successfully demonstrates and compares these different approaches to autonomous navigation.