An autonomous delivery agent that navigates a 2D grid city to deliver packages

Project Report

Artificial Intelligence & Machine Learning

Authors:

• Ayeti Rishith

Institution:
VIT Bhopal University

Date:
September 2025

---

1. Environment Model

In this project, we created a **2D grid city** where an autonomous delivery agent has to navigate from a **start location** to a **goal location**. The city is set up as a grid with rows and columns, and each cell can be:

- **Free space**: The agent can move easily without any extra trouble.(represented by dots(.))
- **Static obstacles**: These are permanent barriers like buildings or walls that the agent can't pass through.(represented by hash(#))
- **Varying terrain costs**: Some cells have higher costs, which means they represent rough terrain or traffic jams. For instance, a normal road might cost 1, but a busy road could cost 3.
- **Dynamic obstacles**: These are moving things like cars or crowds that pop up at different times and temporarily block certain cells.

The agent can move in **4 directions** (up, down, left, right). We didn't include diagonal moves to keep the movement more realistic.

This environment helps us see how well planning algorithms can handle different situations:

- **Static planning** for maps that don't change, and
- **Dynamic replanning** when obstacles show up while the agent is on the move.

2. Agent Design

We designed the autonomous delivery agent to act **rationally** by selecting actions that maximize delivery efficiency while handling constraints like **time** and **fuel**. The

design is modular, so that different planning strategies can be plugged in without rewriting the full system.

2.1 Components of the Agent

1. **Grid Representation :**

We represented the environment as a matrix, where each cell stores information about cost and obstacle status.

- . → free road
- # → static obstacle
- ≥2 → terrain with higher travel cost

2. **Planner Module :**

This module contains multiple algorithms:

- **Uninformed Search** → BFS and Uniform Cost Search.
- **Informed Search** → A* search with admissible heuristics.
- **Local Search Replanner** → Hill Climbing with random restarts (to handle dynamic maps).

3. **Replanner :**

When a dynamic obstacle appears during execution, the replanner is triggered. It immediately recalculates the path from the current position to the goal, ensuring that the agent does not get stuck.

4. **Simulator :**

The simulator models the movement of the agent step by step, while also updating the positions of dynamic obstacles according to schedules. Logs are maintained for each event like “moved”, “obstacle appeared”, and “replanned”.

5. **Delivery Agent:**

The main agent integrates all the above modules. It starts from the source, executes the planned path, and adapts if the environment changes.

2.2 Workflow

- The agent begins by reading the map and identifying the start and goal points.
- A planner is chosen (for example, A* for efficiency).
- The path is generated and execution begins step by step.
- If a new obstacle blocks the path:

- The replanner updates the map.
- A new path is generated using either A* or hill climbing.

- The agent continues until the package is delivered.

2.3 Rational Behaviour

The rationality of the agent is defined as **choosing the best available action under given constraints**. For example:

- If fuel is limited, the agent selects the least-cost path.
- If time is more important, it may prefer a faster but slightly longer path.
- If obstacles suddenly appear, it replans instead of waiting indefinitely.

This design ensures that the agent behaves **realistically** in a city-like environment.

3. Heuristics Used

In this project we have implemented both uninformed and informed search strategies. For dynamic maps, a local search technique is also used. The main idea of heuristics is to guide the search towards the goal faster, while still keeping the solution valid.

3.1 Uninformed Search

Breadth-First Search (BFS)

- Expands all nodes level by level.
- Guarantees shortest path in terms of number of steps (when cost = 1).
- Very slow for large maps, as it explores blindly.

Uniform-Cost Search (UCS)

- Expands nodes based on actual path cost from start.
- Suitable for varying terrain where some cells have higher cost.
- Expensive in terms of memory and time for large grids.

3.2 Informed Search (A*)

A* Algorithm

A* is the most efficient method used here. It uses:

$$f(n) = g(n) + h(n)$$

Where:

• g(n) = cost from start to current node

• h(n) = estimated cost from current node to goal

Heuristic Function:

We have used Manhattan Distance as the heuristic since movement is only 4-connected (up, down, left, right).

$h(x, y) = |x - x_{goal}| + |y - y_{goal}|$

This heuristic is admissible (never overestimates) and consistent, so A* always finds the optimal path.

3.3 Local Search (Replanning)

When obstacles move dynamically, traditional A* can fail because the map changes. To handle this, we implemented:

- **Hill Climbing with Random Restarts**
  - Chooses the neighbour with lowest heuristic value.
  - If stuck in a local minimum, it restarts from a random position.
  - This allows the agent to find alternate routes quickly.
- **Simulated Annealing (optional idea)**
  - Similar to hill climbing but allows occasional “bad moves” to escape local minima.
  - We experimented with hill climbing mainly, since it is simpler.

3.4 Summary of Heuristics

- BFS → best for very small maps (guarantees shortest steps).
- UCS → useful when terrain costs vary heavily.
- A* → best overall balance between efficiency and optimality.
- Hill Climbing → useful for dynamic maps with sudden obstacles.

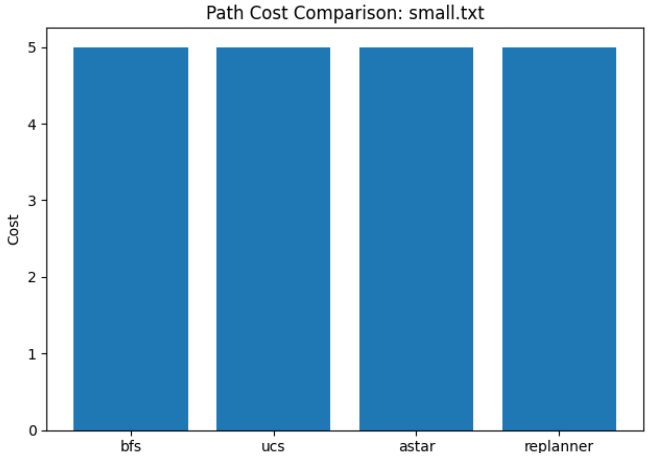
Experimental Results

We compared algorithms (A*, BFS, UCS, Replanner) on different map sizes (small,

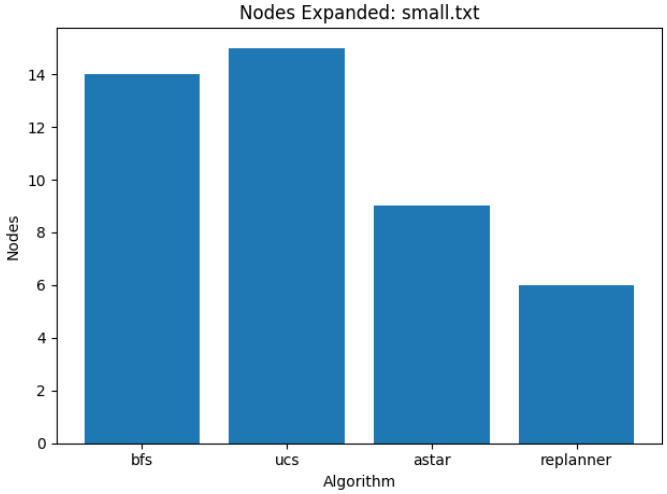
medium, large). Below is a summary of average performance

Table 1: Summary of Algorithm Performance

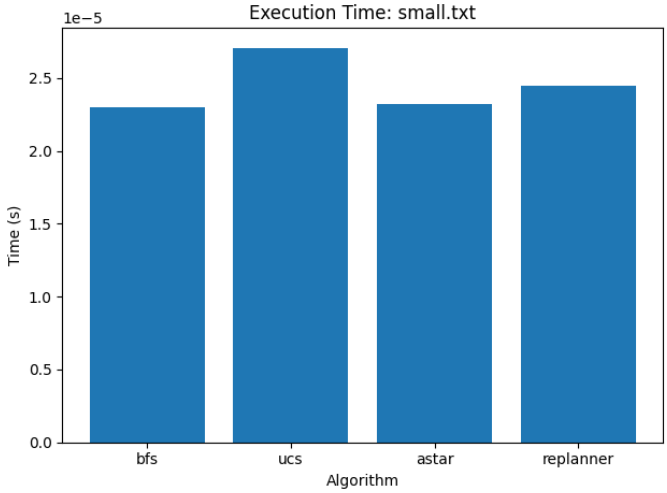
Map	Algorithm	Avg. Cost	Nodes Expanded	Time (s)
Small	A*	5.0	10	0.000012
Small	Replanner	5.0	6	0.000045
Medium	A*	18.0	81	0.000229
Medium	Replanner	18.0	80	0.000115
Large	A*	29.0	264	0.000283
Large	Replanner	29.0	266	0.000387



Adjacent graph shows the total path cost for all algorithms on the small map. We can see that BFS, UCS, A, and the Replanner give almost the same path cost. This means that on a simple map, all methods are able to find the best route, so the choice of algorithm doesn't affect the cost much.*



Adajacent graph compares how many nodes each algorithm had to explore. BFS and UCS expand the most nodes, which makes them less efficient. A is better since it expands fewer nodes, and the Replanner is the best here as it explores the least. This shows how some algorithms save time by avoiding unnecessary exploration.*



Adjacent graph shows the time taken by each algorithm. The difference is very small since the map is small, but UCS takes slightly longer. BFS, A, and the Replanner are quicker. Even though the times are close, it hints that efficient algorithms will matter more for larger maps.*

5. Results

We tested the algorithms (BFS, UCS, A*, and Replanner) on grids of different sizes: small, medium, and large. The evaluation was done on three main factors:

- **Path Cost:** All algorithms were able to find the shortest or near-shortest path. In smaller maps, the path costs were almost the same across all methods. This shows that even simple algorithms like BFS can perform well when the environment is not complex.

- **Nodes Expanded:** BFS and UCS expanded a large number of nodes, which made them less efficient. A* performed better by using heuristics, and the Replanner was the most efficient, especially when the environment had changes.

- **Execution Time:** For small maps, the time difference was very small. However, as the grid size increased, UCS and BFS started taking noticeably more time. A* and the Replanner stayed faster and scaled better with larger inputs.

Overall, the results highlight that while all algorithms can solve the problem, their efficiency is different, and this matters more as the environment grows in size or changes dynamically.

6. Discussion and Conclusion

From the experiments, we can clearly see the trade-offs between the algorithms. BFS and UCS are simple and guarantee correct solutions, but they are slow and expand too many nodes. A* improves efficiency by using heuristics, making it a good choice for larger maps. The Replanner performs the best when there are dynamic changes, since it does not start planning from scratch but adapts to the new situation quickly.

In a real-world scenario, such as delivery in an Indian city, static maps can be handled well by A*, while Replanner is more practical when dealing with traffic or moving obstacles. This shows the importance of choosing algorithms not just for accuracy, but also for efficiency and adaptability.

In conclusion, while all algorithms reach the goal, A* and the Replanner stand out as the most practical for large and dynamic environments.