Autonomous Delivery Agent in a 2D Grid Environment

Submitted by-

Darsh Shukla

24BAS10080

VIT Bhopal University

Abstract

This project presents the design of an autonomous delivery agent that works in a 2D grid-based environment. The agent is responsible for picking up and delivering packages while managing fuel and time limits. The environment is designed with static obstacles, different terrain costs, and dynamic moving obstacles that require the agent to replan its path when conditions change. To achieve this, several algorithms are implemented, including Breadth-First Search (BFS), Uniform Cost Search (UCS), A* with an admissible heuristic, Hill Climbing with restarts, and Simulated Annealing. A graphical interface built using Tkinter allows real-time visualization of the agent's movement and performance comparison of algorithms across different map sizes. Experimental tests were carried out on small, medium, large, and dynamic maps to measure path cost, nodes expanded, computation time, and delivery success. The results show that while A* provides a good balance between efficiency and solution quality, BFS and UCS guarantee optimal solutions but are slower on larger maps. Local search methods work faster in dynamic conditions but do not always produce the most optimal results.

Introduction

- Autonomous delivery systems are becoming increasingly important in areas such as logistics, robotics, and smart cities.
- The goal of this project is to design an autonomous delivery agent that can operate in a 2D grid environment.
- The environment includes:
  - Static obstacles (buildings, blocked paths)
  - Varying terrain costs (different movement difficulties)
  - Dynamic obstacles (vehicles or moving objects)
- The delivery agent must:
  - Pick up and deliver packages
  - Work under fuel and time constraints
  - Replan routes when conditions change
- To solve this, the project uses classical AI search algorithms:
  - Uninformed search: Breadth-First Search (BFS), Uniform Cost Search (UCS)
  - Informed search: A* (with heuristic)
  - Local search: Hill Climbing, Simulated Annealing
- A Tkinter-based GUI is included for:
  - Visualization of the grid, agent, and obstacles
  - Real-time display of package delivery progress
  - Comparison of algorithm performance across maps
- Significance:
  - Demonstrates how AI algorithms perform under different constraints
  - Provides insights into optimality vs speed vs adaptability trade-offs
  - Serves as a practical example of AI applied to real-world inspired delivery problems

Environment & Agent Design

Environment Design

- The project environment is modeled as a **2D grid** where each cell represents a location in the city.
- Each grid cell has:
    - **Terrain cost** (movement difficulty, ≥ 1)
    - **Obstacle status** (blocked or free)
- The environment includes different types of challenges:
    - **Static obstacles** – fixed blocked cells (e.g., buildings).
    - **Varying terrain costs** – some paths take more fuel/time to cross.
    - **Dynamic moving obstacles** – vehicles moving across cells according to schedules.
- Multiple map configurations are used:
    - **Small Map (5x5)** – basic obstacles.
    - **Medium Map (10x10)** – higher terrain costs and more obstacles.
    - **Large Map (10x10)** – dense obstacles.
    - **Dynamic Map (8x8)** – moving vehicles requiring replanning.

Agent Design

- The **Delivery Agent** starts at a fixed position with:
    - **Fuel limit** – every move reduces available fuel.
    - **Time limit** – prevents infinite running.
    - **Package list** – pickup and drop-off locations.
- The agent's tasks:

1. Navigate to the package location.
2. Pick up the package.
3. Plan and execute a route to the delivery location.
4. Repeat until all packages are delivered or resources are exhausted.

- Key features:
    - **Dynamic Replanning** – if the chosen path gets blocked by a moving obstacle, the agent replans.
    - **Logging system** – records steps, nodes expanded, and delivery progress.
    - **Graphical Visualization** – Tkinter interface shows agent position, obstacles, packages, and delivery status in real-time.

Algorithms

Algorithms Implemented

The project uses a variety of classical AI search algorithms to plan efficient paths for the delivery agent:

1. Uninformed Search

- **Breadth-First Search (BFS):**
    - Expands nodes level by level.
    - Complete and guarantees the shortest path in unweighted grids.
    - Inefficient for large maps as it explores all possibilities.
- **Uniform Cost Search (UCS):**
    - Expands the lowest-cost node first.
    - Finds the optimal path when movement costs vary.
    - More expensive in terms of computation than BFS.

2. Informed Search

- **A* Search:**
    - Uses both actual cost (g) and heuristic estimate (h).
    - Heuristic: Manhattan distance (grid-based distance).
    - Provides a good balance between efficiency and solution quality.

3. Local Search

- **Hill Climbing:**
    - Greedy algorithm, always moves toward the best local option.
    - Fast but may get stuck in local minima.
    - Random restarts help escape dead ends.
- **Simulated Annealing:**
    - Similar to hill climbing but accepts worse moves with some probability.
    - Probability decreases with a cooling schedule.
    - Allows exploration of wider solution space and avoids local minima.
    -

Algorithm Comparison

Algorithm	Completeness	Optimality	Best Use Case
BFS	Yes	Yes	Small maps, unweighted environments
UCS	Yes	Yes	Cost-sensitive maps with varying terrain
A*	Yes	Yes	Balanced performance, medium/large maps
Hill Climbing	No	No	Quick approximate solutions
Sim. Annealing	No	Sometimes	Large, dynamic maps with many obstacles

Experimental Setup & Results

Experimental Setup

- The agent was tested on four different map environments:
    1. **Small Map (5x5):** Basic obstacles, easy navigation.

2. **Medium Map (10x10):** Higher terrain costs and scattered obstacles.
3. **Large Map (10x10):** Denser obstacles with multiple routes.
4. **Dynamic Map (8x8):** Moving obstacles requiring replanning.

- **Evaluation Metrics**:
  - Path Cost: Total cost of reaching the goal.
  - Nodes Expanded: Number of explored states.
  - Computation Time: Time taken by the algorithm.
  - Delivery Success: Whether all packages were delivered.
- **Environment Constraints:**
  - Limited fuel (1000 units).
  - Limited time (1000 steps).
  - Packages with fixed pickup and delivery points.

Results

Map	Algorithm	Success	Path Cost	Nodes Expanded
Small	BFS	Yes	Low	High
Small	UCS	Yes	Low	Medium
Small	A*	Yes	Low	Low
Small	HillClimb	Yes	Medium	Low
Small	SimAnneal	Yes	Medium	Low
Medium	BFS	Yes	Medium	Very High
Medium	UCS	Yes	Medium	High
Medium	A*	Yes	Medium	Low
Large	BFS	Yes	High	Very High
Large	UCS	Yes	High	High

Large	A*	Yes	High	Medium
Dynamic	A*	Yes	Medium	Medium
Dynamic	HillClimb	Yes	Medium	Low
Dynamic	SimAnneal	Yes	Medium	Low

Observations

- **BFS** works best for very small maps but becomes slow as the map size grows.
- **UCS** guarantees optimal cost but explores more nodes than A*.
- **A*** consistently balances **efficiency and solution quality**, making it the most reliable for medium/large maps.
- **Hill Climbing** is fast but may miss optimal paths. Random restarts help improve success rate.
- **Simulated Annealing** performs well in dynamic scenarios where flexibility is more important than exact optimality.

Analysis

I. **Algorithm Trade-offs:**

- **BFS/UCS** → always find the optimal path but become very slow on larger or more complex maps.
- **A*** → balances optimality and speed, making it the most reliable choice overall.
- **Hill Climbing** → fast in execution but can get stuck in local optima; works better with random restarts.
- **Simulated Annealing** → flexible in dynamic conditions, able to escape local traps, but results vary between runs.

II. **Performance Insights:**

- On **small maps**, all algorithms perform well with little difference.
- On **medium/large maps**, A* is clearly more efficient than BFS/UCS.

- On **dynamic maps**, local search (Hill Climbing, Simulated Annealing) handles unexpected changes better.
- GUI visualization slows down comparisons but helps demonstrate the agent's behavior effectively.

Conclusion

This project successfully demonstrates the design and implementation of an autonomous delivery agent operating in a 2D grid environment. By incorporating multiple AI search algorithms, the agent was able to navigate static and dynamic obstacles, manage fuel and time limits, and deliver packages efficiently. The experimental results highlight the strengths and weaknesses of each algorithm, showing that A* provides the best balance for most cases, while local search methods are more adaptable in dynamic situations.

The project also developed a Tkinter-based GUI, which allows users to visualize the environment, track the agent's movements, and compare algorithm performance interactively. This integration of algorithms, simulation, and visualization makes the project both practical and educational.

Future Work

- Extend to **multi-agent delivery systems** for collaboration and competition.
- Introduce **probabilistic or stochastic obstacles** instead of fixed schedules.
- Optimize the GUI to allow faster "Compare All" runs without visualization delays.
- Explore more advanced heuristics (e.g., Euclidean distance, weighted A*) for better efficiency.
- Scale the environment to larger grids and more complex city layouts.