HiLCoE School of Computer Science and Technology

Artificial Intelligence(CS488) Group Assignment: Addis Ababa Map Route

Group Members:

- Alazar Dula(DH2573)
- Estifanos Behailu(DQ8168)
- Kidus Mesfin(WE6738)
- Kebron Yohannes(JL7957)
- Luliam Tekle(OD6165)

Date:- November 20th, 2025

1. Problem Understanding

The task we have at hand is to build a system that finds the optimal path between two locations within Addis Ababa's city map, represented as a weighted graph. The system supports UCS and A* Search algorithm, allowing users to build their own graph, enter 'START' and 'GOAL' locations, receive optimal path and cost, and avoid repeatable state expansions.

2. Algorithm Descriptions

Uniform Cost Search (UCS) Algorithm

UCS is a path-finding that expands the least cost mode first, ensuring that the path to the goal node has the minimum cost.

A* Search Algorithm

A* Search algorithm is an informed Best-First Search algorithm that efficiently determines the lowest cost path between any two nodes in a directed weighted graph with non-negative edge weights.

Advantages and Limitations

Algorithms	Strengths	Limitations
UCS	Optimal + simple	Slow with heuristics
A*	Optimal + good with faster heuristic	Requires heuristic design

3. Implementation Overview

In the program, we imported 'heapq' to use the priority queue for UCS and A* Search. Both algorithms used a **closed list** to avoid revisiting nodes. Python dictionaries is used for the graph adjacent list and used user input for dynamic graph creation, heuristic, and algorithm selection.

The modular functions used in Python includes ***get_graph_from_user()***, ***get_heuristic_from_user()***, ***uniform_cost_search()***, and ***a_star()***.

For the sake of testing and checking our code we have also added a similar implementation but with a defined graph and heuristic that takes input of start and destination. The uncommented part is for the graph provided and if a new graph is needed you can simply uncomment the above part and make a new graph for testing.

Example snippet (UCS priority queue):

```
pq = [(0, start, [])]
cost, node, path = heapq.heappop(pq)
```

4. Experimental Results

Algorithm	Path	Cost	Nodes Explored
UCS	Aratkilo -> Megenagna -> Gerji	385	3
A*	Aratkilo -> Megenagna -> Gerji	385	3

We were able to locate a small error on the heuristic table on the part that says Aratkilo to Gerji it says the value should be 510 but actually based on the graph it should be 385. No matter the code checks heuristic on the nodes so prior to movement so it didn't affect the A* search and most of the outputs for both are bound to be the same..

5. Analysis and Comparison

- The UCS found the correct path but explored more nodes.
- A* performed faster due to the heuristic values based on the Addis Ababa distance table.
- For well-structured heuristics, A* is noticeably more efficient.

6. Reflection on Learning and How AI Tools Were Used

We had made use of the genrative AI tools during the developmental phase such as ChatGPT and Gemini. Implementing the algorithms was one of the most challenging parts when building the system.

How Was AI Tools Used?

- **Debugging:** The AI tools helped in identifying the logical mistakes in queue handling and closed list implementation.
- **Modularity:** It suggested the restructuring of the program into reusable functions (graph input, heuristic input, UCS, A*).
- **Input Structuring:** The AI tools helped in designing simple, user-friendly input formats for graph edges and heuristics.
- **Commenting:** It assisted in adding consistent and clear comments across the file.
- **Algorithm Clarity:** The AI tools provided explanations in comparing UCS, A* Search, and Dijkstra Algorithm to strengthen understanding.

What We Did Ourselves?

- Implemented both algorithms, tested all cases, and designed the final input/output structure.
- Modified AI suggestions to fit our assignment requirements.
- Expand all logic and implementation choices were aligned with the course materials.

7. Conclusion

Working on this assignment helped us understand more on what the graph search algorithm is, how the heuristics is designed, and what modular programming is all about. It also gave us a better understanding on how to implement heuristic functions and values alongside the shortest path problems where we normally would have used BFS or Djestra.