Kalinga Institute of Industrial Technology

Deemed to be University

Lab Mini Project 2

Shortest Path using Uniform Cost Search

Aman Pathak	22051662
Abhinandan Maji	2205784
Aditya Datta	22051223
Chayan Bera	22051506
Devi Prasad Panda	22051511
Garv Agarwal	22051159
Rachit Raj Krishna	2205054
Purbasha Nayak	22051712

supervised by

Dr. Sambit Praharaj

January 24, 2025

1 Introduction to Uniform Cost Search (UCS)

Uniform Cost Search (UCS) is a fundamental algorithm in artificial intelligence and graph theory used to find the shortest path between nodes in a weighted graph. Unlike breadth-first search (BFS), UCS considers edge costs and expands the path with the lowest cumulative cost.

1.1 How UCS Works

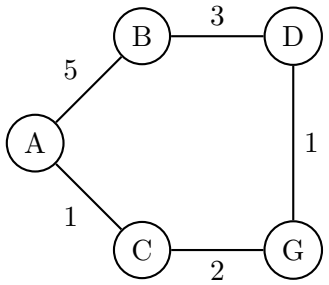

Figure 1: Example graph for UCS demonstration (Numbers represent edge costs)

1.1.1 Algorithm Steps

Step 1: Initialize priority queue with start node (cost = 0)

Step 2: While queue is not empty:

- Dequeue node with **lowest cumulative cost**
- If goal node found, return path
- Expand to neighboring nodes
- Enqueue neighbors with updated path cost

Step 3: If queue empties, return failure

1.2 Complexity Analysis

- **Time Complexity**: $O((E + V) \log V)$
  Where E = edges, V = vertices. Uses priority queue (typically binary heap)
- **Space Complexity**: $O(V)$
  Stores all nodes in worst case

1.3 Optimality and Completeness

- **Optimal**: Always finds least-cost path when:
  - All edge costs are non-negative
  - Path cost increases with depth
- **Complete**: Guaranteed to find solution if one exists

1.4 Comparison with Other Algorithms

- **vs BFS**: UCS generalizes BFS (BFS uses edge count as cost)
- **vs Dijkstra's**: Essentially identical - both find shortest paths
- **vs A***: A* uses heuristics to guide search, UCS is heuristic-free

2 Implementation

This code implements the Uniform Cost Search (UCS) algorithm to find the shortest path between two nodes in a weighted graph. UCS is a graph traversal algorithm that finds the path with the lowest cumulative cost from a starting node to a goal node.

2.1 Code Structure

The core of the implementation is the `uniform_cost_search(graph, start, goal)` function:

- **Input:** A graph `graph`, a starting node `start`, and a goal node `goal`.
- **Data Structures:**
  - `priority_queue`: A min-heap (implemented using `heapq`) storing tuples of `(cumulative_cost, current_node, path)`.
  - `visited`: A dictionary storing the minimum cost to reach each visited node.
- **Algorithm:**
  1. Initialize the `priority_queue` with the starting node and a cost of 0.
  2. While the `priority_queue` is not empty:
     (a) Pop the node with the lowest cumulative cost.
     (b) If the current node is the goal, construct the path and return it along with the total cost. Color the path edges green in the graph.

(c) If the current node has been visited with a lower cost, skip it.

(d) For each neighbor of the current node:

i. Calculate the cost to reach the neighbor.

ii. If the neighbor is unvisited or the new cost is lower than the previously recorded cost, add it to the `priority_queue`.

3. If the goal is not reachable, return `None` and infinity.

- **Output:** A tuple containing the path (list of nodes), the total cost, and the graph with colored path edges.

2.2 Key Concepts

- **Priority Queue:** Ensures that nodes with the lowest cost are explored first.
- **Visited Set:** Prevents revisiting nodes along suboptimal paths, improving efficiency.
- **Graph Representation:** Uses `networkx` to represent the graph, allowing for weighted edges.

This implementation provides an efficient way to find the shortest path in a weighted graph using the Uniform Cost Search algorithm.

2.3 Output Interpretation

Table 1: Search Algorithm Comparison

Nodes	UCS		BFS		DFS	
	Time (s)	Path Length	Time (s)	Path Length	Time (s)	Path Length
50	0.001392	4	0.001102	4	0.000878	4
100	0.001709	6	0.001733	4	0.001799	23
200	0.004672	6	0.003726	6	0.003607	52
400	0.005482	0	0.005958	0	0.005479	0
800	0.014376	8	0.015159	7	0.018944	424
1000	0.023005	8	0.018543	6	0.021258	412
1500	0.041061	12	0.028889	8	0.062434	862

3 Applications

- Network routing protocols
- GPS navigation systems

- Robotics path planning
- Puzzle solving (e.g., sliding tile puzzles)

4 Conclusion

Uniform Cost Search provides an optimal solution for pathfinding in weighted graphs with non-negative edge costs. While not as fast as heuristic-based methods like A* for many problems, it remains fundamental in AI due to its guaranteed optimality and simplicity.