D795 - Task 1 - Attempt 1

Silver Alcid - #011933973

8/07/2025

B1. Discuss the overall operation of the application.

The ambulance dispatch system runs a simulation using input CSV files to model ambulance dispatching in response to emergency calls.

Initialization Phase:

- Load network topology (location_network.csv), ambulance data (ambulance.csv), priority rules (call_priority.csv), and incoming calls (calls.csv).
- Construct necessary data structures, such as graphs for locations and queues for managing calls.

Dispatch Phase:

- Iterate through the list of emergency calls.
- Prioritize calls based on defined priority levels and arrival order.
- Calculate the fastest route from every available ambulance's staging location using the specified routing algorithm for each call.
- Select the ambulance with the shortest total time (including delays) for each call.
- Create a dispatch record containing all required details for each call.
- Log the dispatch to ambulance_call_log.csv for each call.
- Reset the dispatched ambulance's location to its original staging location and mark it as available for each call.

Completion Phase:

- Continue processing until all calls are handled.

B2. Delineate how the application will process the simulation files.

location_network.csv:

- Defines a graph of location nodes and weighted edges (travel times).
- Used by routing algorithms to compute fastest paths.

ambulance.csv:

- Lists each ambulance's unique ID and staging location.
- Initializes ambulance availability and base assignments.

call_priority.csv:

- Maps call priorities to required response time thresholds.
- Used for prioritization during dispatching.

calls.csv:

- Contains time-stamped call records with ID, type, location, and priority.
- Drives the simulation loop, feeding the dispatch queue.

B3. Describe how the application will manage call priorities.

Priority Handling:

- Calls are sorted by priority level.
- Within each priority level, calls are processed in arrival order.

Ambulance Selection:

- For each call, calculate fastest route from each available ambulance using the routing algorithm defined in Part C.
- Include delay times in total response time calculation.
- Select the ambulance with the lowest total time to the call location.

Dispatch Logging:

- Create a dispatch record with Call ID, Call Type, Call Location, Selected Ambulance, Route to Call Location, Time to Call Location.
- Format this as name-value pairs and append to ambulance_call_log.csv.
- Reset the dispatched ambulance to its base and mark it available.

C1-C4. Explain the operation of each algorithm that can be used to calculate the fastest route. Describe how each algorithm will be used in the application. Calculate the Big O time complexity and the Big O space complexity of each algorithm. Explain the advantages and disadvantages of each algorithm.

Prototype A: Dijkstra's Algorithm

Operation

Dijkstra's algorithm computes the shortest paths from a source node to all other nodes in a graph with non-negative edge weights. (Goldwasser, Goodrich, & Tamassia, 2024). Ituses a priority queue to always expand the node with the currently known shortest distance. At each iteration, it updates the shortest path estimates to neighboring nodes, repeating until all reachable nodes have been visited or the destination is found.

Use in the Application

For each emergency call, the algorithm calculates the shortest path from every available ambulance's staging location to the call location. The total time (path cost plus any ambulance delay) is computed for each. The ambulance with the lowest overall time is selected, dispatched, and its route and arrival time are recorded in the dispatch log.

Big O Complexity

- Time Complexity: O((V + E) log V) using a binary heap
- Space Complexity: O(V)

Advantages

- Guarantees optimal (shortest) paths in graphs with non-negative weights
- Consistently accurate and reliable for a wide range of graph structures

Disadvantages

- Inefficient for single-destination queries, as it explores paths to all nodes
- May be slower on large or sparse graphs when only one target node is needed

Prototype B: Uniform Cost Search (UCS)

Operation

Uniform Cost Search is a variant of Dijkstra's algorithm that focuses on finding the lowest-cost path from a source to a specific goal. It uses a priority queue to expand the path with the lowest cumulative cost and stops immediately once the goal node is reached.

Use in the Application

For each emergency call, UCS is run from each ambulance's staging location to the call location. The search stops as soon as the goal (call location) is found. The ambulance with the shortest total cost is chosen and its route and response time are recorded in the dispatch log.

Big O Complexity

- Time Complexity: O((V + E) log V) using a binary heap
- Space Complexity: O(V)

Advantages

- More efficient for single-goal problems because it stops early
- Still guarantees the optimal path as long as all edge weights are non-negative

Disadvantages

- Can still explore a large portion of the graph if the goal is far or the graph is dense
- Offers limited performance improvement over Dijkstra's in large, complex graphs where many paths must be evaluated anyway

G1. Execute Prototype One 10 times and calculate and present the average execution time for finding the fastest route algorithm.

Dijkstra's Algorithm Results

Run #	**Result**
1	.0048 seconds
2	.0048 seconds
3	.0046 seconds
4	.0048 seconds
5	.0046 seconds
6	.0045 seconds
7	.0048 seconds
8	.0048 seconds
9	.0048 seconds
10	.0046 seconds
Average	.00471 seconds

G2. Execute Prototype Two 10 times and calculate and present the average execution time for finding the fastest route algorithm.

Uniform Cost Search Results

Run #	Result
1	.0025 seconds
2	.0024 seconds
3	.0024 seconds
4	.0023 seconds
5	.0023 seconds
6	.0022 seconds
7	.0023 seconds
8	.0023 seconds
9	.0023 seconds
10	.0024 seconds
Average	.00234 seconds

H1. Compare the performance of both algorithms.

The performance comparison between Dijkstra's Algorithm and Uniform Cost Search (UCS) in this simulation reveals a consistent and measurable difference in execution efficiency. Across ten runs, Dijkstra's Algorithm demonstrated an average execution time of 0.00471 seconds, while UCS achieved a lower average of 0.00234 seconds. This shows us that UCS is about 50% faster than Dijkstra's Algorithm under identical conditions. Both algorithms are practically similar in terms of exploring the shortest paths in weighted graphs; however, the implementation of UCS in this context appears to benefit from a more optimized or streamlined search process, potentially due to reduced overhead or more efficient use of the priority queue. This suggest that for this specific application and data structure, UCS offers superior performance in computing the fastest route.

H2. Provide one recommendation for improving each algorithm based on your interpretation of the performance testing.

Dijkstra's Algorithm: A simple optimization would be to use Python's heapq directly with a custom priority queue, instead of relying on NetworkX's built-in implementation. This can reduce overhead and give better control over the algorithm's efficiency.

Uniform Cost Search: One improvement could be to avoid reprocessing already visited nodes by using a dictionary to store the lowest known cost to each node. This would help ensure that the algorithm doesn't waste time exploring less efficient paths to the same location.

References

Goldwasser, M., Goodrich, M. T., & Tamassia, R. (2024). Algorithm design and applications. zyBooks. https://learn.zybooks.com/zybook/NavaAlgorithmNov2024