Theoretical Algorithms Analysis Project

Team Members:

- **Abdelrahman khalid akl / 22100270**
- **Hazem khaled kassem / 22101258**
- **Omar Islam Mahmoud / 22100330**
- **Abdelrahman Ragab Marey / 22101325**
- **Mohamed Ahmed Ibrahim /22101115**

Theoretical Algorithms Analysis: Dijkstra’s Algorithm

1. Introduction

Algorithm Name: Dijkstra’s Shortest Path Algorithm (Modified for Traffic & Cost)
Domain: Transportation Optimization
Use Case: Find the most efficient path considering time, distance, and cost with traffic conditions.

2. Mathematical Foundations and Proof of Correctness

Graph Representation: Directed weighted graph G=(V,E) where weights represent cost, time, or distance.

Correctness Principle:

- Relies on the **Greedy Choice Property** and **Optimal Substructure**.
- Once a node’s shortest path is finalized, it cannot be improved.

Proof Sketch:

- Inductive approach showing that after extracting a node from the priority queue, its shortest path weight is correct.
- Based on triangle inequality: for any path u→v→wu \to v \to wu→v→w, d(u,w)≤d(u,v)+d(v,w)d(u, w) \led(u, v) + d(v, w)d(u,w)≤d(u,v)+d(v,w).

3. Pseudocode (Adapted for Your Case)

python

```
def modified_dijkstra_with_cost(graph, start, end, start_time):
    initialize combined_weights, times, costs, distances,
arrival_times, previous
    queue ← [(0, start)]

    while queue not empty:
        current_weight, current_node ← pop queue
        for neighbor in neighbors(current_node):
            travel_time ← calculate_travel_time(current_node,
neighbor, arrival_times[current_node])
            cost ← graph[current_node][neighbor].cost
            distance ← graph[current_node][neighbor].distance
            weight ← 0.7 * travel_time + 0.3 * cost

            if new_weight < combined_weights[neighbor]:
                update all values
                push (new_weight, neighbor) into queue

    reconstruct path from 'end' using 'previous'
```

4. Time and Space Complexity

- **Time Complexity (Classic Dijkstra with min-heap):**
  O((V+E)logV)O((V + E) \log V)O((V+E)logV)
- **In Your Case (with traffic calculation):**
  - Still O((V+E)logV)O((V + E) \log V)O((V+E)logV) for the core logic
  - Extra cost for `calculate_travel_time`, but it's O(1) per edge (lookup + simple math).
- **Space Complexity:**
  O(V+E) for storing graph, distances, paths.

5. Comparison with Alternative Approaches

Algorithm	Time Complexity	Handles Dynamic Costs	Notes
Dijkstra	O((V+E)logV)O((V + E) \log V)O((V+E)logV)	(with extension)	Efficient, deterministic
A*	Similar to Dijkstra	(if heuristic valid)	Needs good heuristic
Bellman-Ford	O(VE)O(VE)O(VE)		Slower, handles negative weights
BFS	O(V+E)O(V + E)O(V+E)		Only for unweighted graphs

6. Specific Modifications in Your Project

Traffic-Aware Travel Time: Adjusted based on hourly traffic data.

Cost-Weighted Decision Making: Combined travel time and cost using weighted sum.

Arrival Time Tracking: Time-dependent edge weights based on actual travel period.

7. Performance and Optimization

Strengths:

- Realistic traffic modeling.
- Easy to adjust weight coefficients (e.g., 70% time, 30% cost).

Weaknesses:

- No prediction of future traffic states (static per period).
- Assumes deterministic travel times.

Optimization Opportunities:

- Use A* with spatial heuristics (e.g., Euclidean distance).
- Real-time traffic updates.
- Parallel path computation (GPU acceleration).

8. Conclusion and Lessons Learned

Dijkstra's algorithm, even when extended, remains robust and adaptable.
Integrating domain-specific parameters (like traffic) gives more practical results.
Balancing multiple criteria (cost/time) is essential in real-world logistics.
Future enhancements could focus on dynamic updates and predictive modeling.