Adaptive Route Optimization with InTandem

Anh Huynh

Problem Statement & Real-World Use Case

Shipping routes today are mostly static, even when real-world disruptions (e.g., traffic, storms, customs delays) arise; this leads to delayed or costly deliveries. **InTandem** simulates a smarter algorithm that adjusts in real time, re-routing based on live disruption data to optimize delivery time, cost, and risk.

Test Plan & Results

Test 1 – *No Disruptions*

Graph: A→B→C (cost 5), A→C (cost 12)

→ Expected: 5

→ Actual: 5

Test 2 – *Moderate Traffic Disruption on B→C*

Disruptions: {10, 4, 0, 0}

→ Expected: ~9

→ Actual: 9

Test 3 – *Extreme Disruption Avoidance*

B→C highly penalized, A→D→C safer

→ Expected: 6

→ Actual: 6

Test 4 – *Live-Randomized Inputs*

Disruptions generated at runtime

→ Output varies dynamically with penalties applied.

Output:

```
------------------------------
        Test 1: Base Shortest Path without Disruptions
Expected shortest cost to C: 5 (A->B->C: 2+3)
Actual shortest cost to C: 5
Correctly routes A->B->C (5) instead of A->C(12)
------------------------------
        Test 2: Weighted Disruption Penalty Influences Routing
Expected shortest cost to C: 9 (2 + adjusted ~7)
Actual shortest cost to C: 9
Correctly routed A->B->C (~9) instead of A->C (12)
------------------------------
        Test 3: Extreme Disruption Only Affects B->C
Expected shortest cost to C: 6 (A->D->C)
Actual shortest cost to C: 6
Correctly routed around high-penalty B->C path via A -> D -> C
------------------------------
        Test 4: Live Randomized Disruptions at Node C
Generated disruptions for B → C:
Traffic: 3
Weather: 10
Fuel:    5
Customs: 4
Shortest cost to C: 5
```

Core Algorithm

```
for (auto& edge : graph[node]) {
            int adjusted = edge.baseWeight;

            // calculate weighted disruption score for this specific edge
            if (!edge.disruptions.empty()) {
                double weightedScore = 0.0;
                for (size_t i = 0; i < weights.size() && i < edge.disruptions.size(); ++i) {
                    weightedScore += weights[i] * edge.disruptions[i];
                }
                if (weightedScore > 5) {
                    // if the combined disruption score exceeds the threshold (5),
                    // apply a quadratic penalty to the base cost to reflect severity
                    adjusted = static_cast<int>(round(edge.baseWeight * (1 +
pow(weightedScore - 5, 2))));
                }
            }

            // update shortest path if this new adjusted cost is better
            if (cost + adjusted < dist[edge.to]) {
```

```
                dist[edge.to] = cost + adjusted;
                pq.push({dist[edge.to], edge.to});
            }
        }
```

Runtime & Memory

Modified Dijkstra's algorithm.

Time Complexity:

- O((V + E) log V) using a min-heap priority queue, where:
- V = number of nodes (locations)
- E = number of edges (shipping routes)
- O(1) - assuming a fixed number of disruption categories

Space Complexity:

- O(V) for distance map dist
- O(E) storing graph adjacency list
- O(V) priority queue
- Total = O(V + E)

Discussion: Tradeoffs & Future Work

All disruption types are merged into a single score, which simplifies logic but loses specificity. It doesn't consider route-specific thresholds, per-shipment urgency, or multi-objective goals (fastest *vs* cheapest).

Future work includes:

- Time-window constraints
- Multi-shipment load balancing
- Scaling to 100+ node networks
- Real API hooks for live event data
- Route-level disruption customization

Github Repo

https://github.com/anhpls/intandem

Release Tag: v1.0-final

Link to Presentation

https://docs.google.com/presentation/d/1q52GqUrGiFJkLEI7kY9xVXLBqCAOHKBOmbI4TnKsZoM/edit?usp=sharing