

Project Proposal: Algorithm Design and Implementation

1. Team Members

- Joud Kayyali (3230601030)
-

2. Problem Statement

This project aims to design, implement, and analyze an algorithmic solution for a real-world logistics challenge: **Optimal Route Planning for a Delivery Service.**

2.1. Core Problem (Phase 1 & Phase 2 Scope)

The primary objective is to determine the optimal path from a single starting point (e.g., Warehouse “A”) to a single destination (e.g., Customer “B”) within a city grid.

The city is modeled as a weighted, directed graph, where intersections represent vertices and roads represent edges.

The definition of “optimal” is user-dependent and includes the following criteria:

- Least Time:** Calculated using average travel times and current traffic data (highest priority).
- Least Distance:** The shortest physical path.
- Least Cost:** A combined metric using distance and fuel consumption.

2.2. Advanced Problem (Phase 3 & Optimization Scope)

The secondary problem, which will be the focus of Phase 3, is the **multi-stop delivery scenario** (e.g., Start at “A”, visit “B”, “C”, and “D”, then return to “A”).

This represents a simplified variant of the **Traveling Salesperson Problem (TSP)**. The goal is not to fully solve TSP (an NP-hard problem), but rather to use it as a benchmark for evaluating advanced optimization strategies.

3. Literature Review Summary

To address the project's objectives, several shortest-path and graph traversal algorithms were reviewed:

- **Graph Representation:**

The road network will be modeled using an **Adjacency List**, which is efficient for sparse graphs and allows fast neighbor lookups.

- **BFS (Breadth-First Search) & DFS (Depth-First Search):**

Analysis: BFS and DFS are foundational traversal algorithms but unsuitable for our core problem because they operate on **unweighted graphs**.

Limitation: BFS identifies the path with the **fewest edges**, not the one with the least time or distance. It cannot incorporate weighted metrics such as **traffic or varying road lengths**.

Example: A path with 2 long roads (1 hour) might be mistakenly preferred over 3 short roads (20 minutes), which is inappropriate for the logistics domain.

- **Dijkstra's Algorithm (Baseline Approach):**

Analysis: Dijkstra's algorithm is the standard solution for the single-source shortest path problem in **weighted graphs** with **non-negative edge weights**.

Application: This algorithm is an ideal fit for the Core Problem, as the weights (time, distance, or cost) are always non-negative. It will serve as **our baseline implementation in Phase 2**.

- **A* Search Algorithm (Phase 3 Optimization):**

Analysis: A* is an informed search technique that **extends Dijkstra** by introducing a **heuristic function** to estimate the remaining distance or cost to the goal. This allows for faster exploration by prioritizing more promising paths.

Application: In Phase 3, we will implement A* and compare its performance against Dijkstra, especially on large-scale maps.

- **Bellman-Ford Algorithm (Addressing Limitations):**

Analysis: A key limitation of Dijkstra's algorithm is its inability to **handle negative weights**.

Limitation Example: In the case of an electric vehicle (EV) traveling downhill, the battery may regenerate energy (e.g., -2% energy cost), which is modeled as a "negative weight."

Application: Phase 3 will include an analysis of the Bellman-Ford algorithm, which supports negative weights. We will evaluate trade-offs between:

- **Dijkstra:** Fast but incompatible with negative weights.
- **Bellman-Ford:** Slower but capable of modeling more complex real-world scenarios.

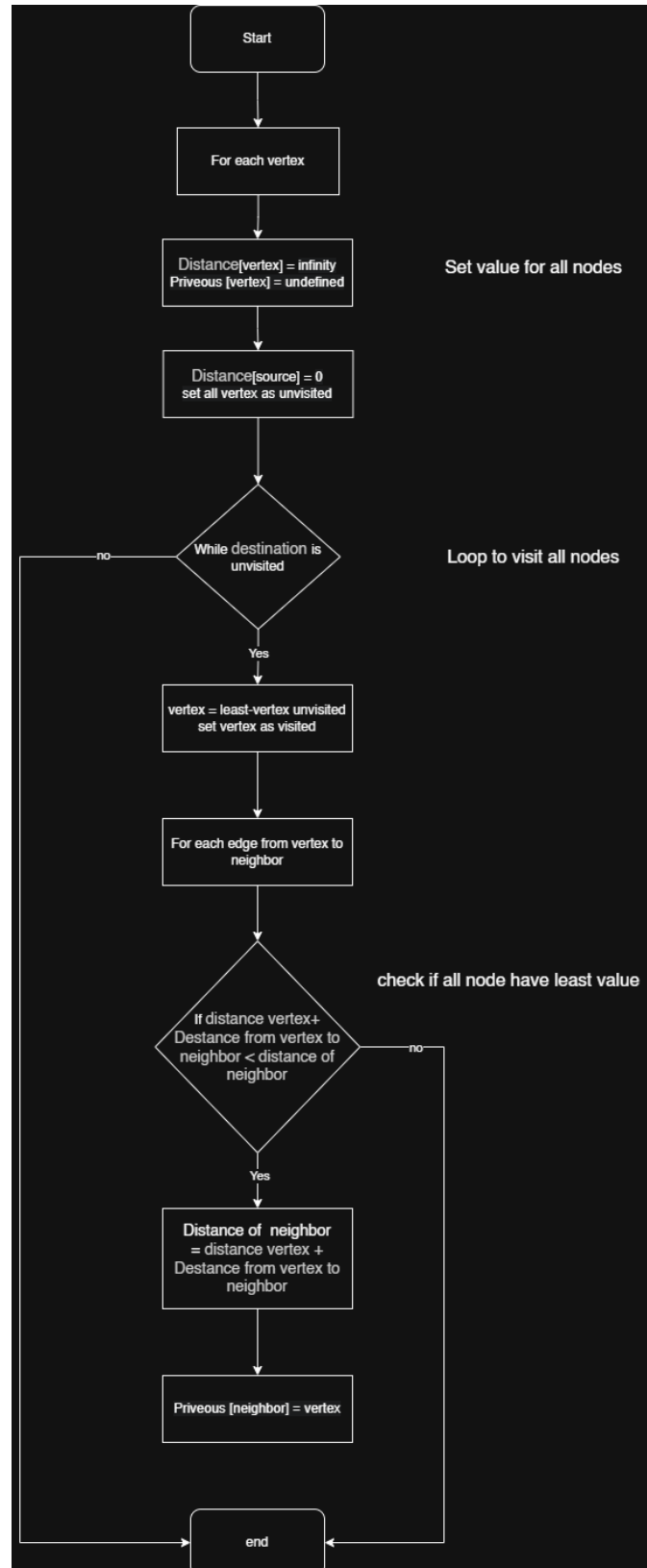
4. Proposed Algorithm Design (Phase 1)

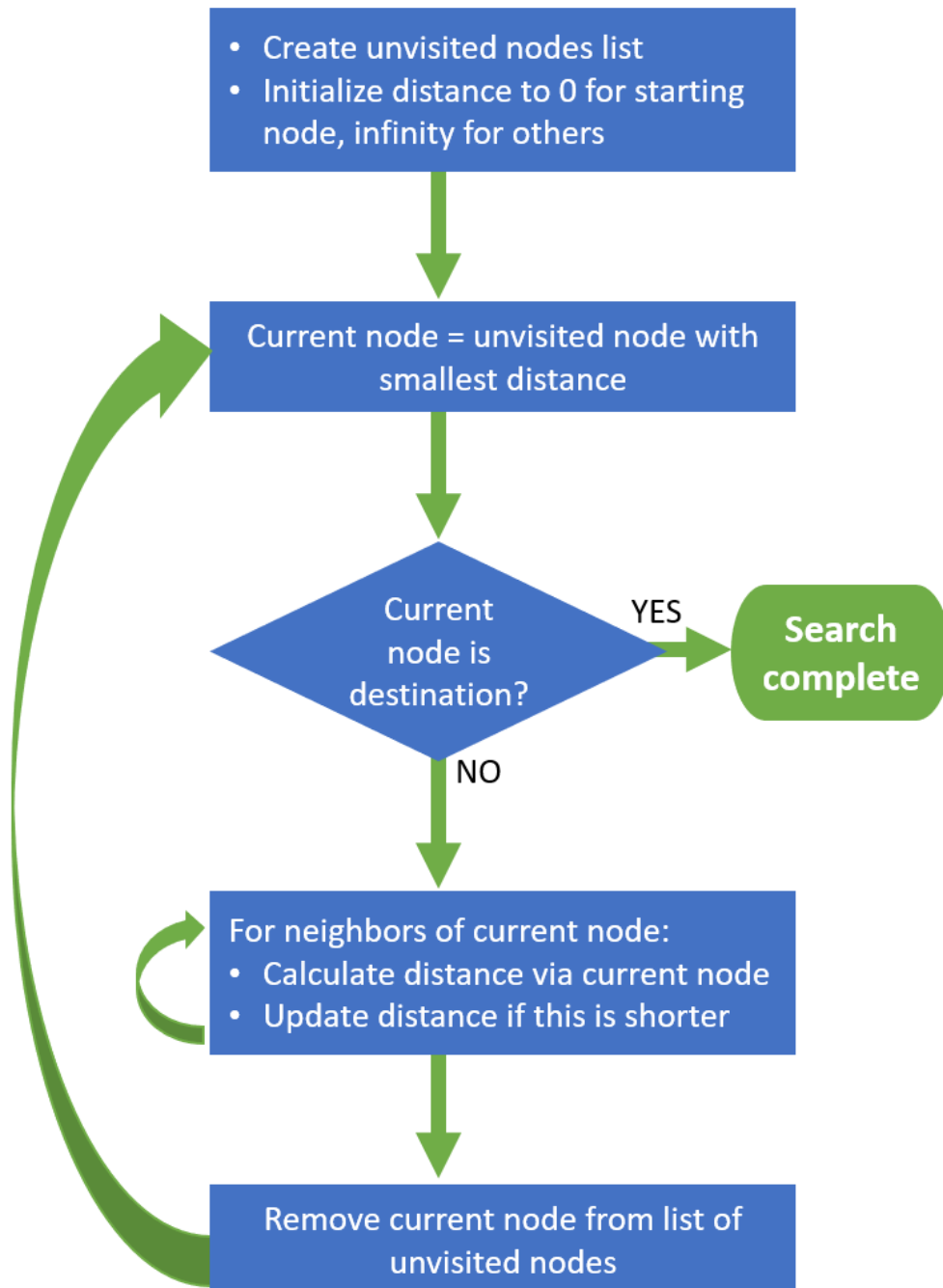
For the initial implementation (Phase 2), the proposed algorithm is **Dijkstra's Algorithm**, as it effectively solves the *Core Problem* and provides a reliable baseline for later optimization and performance comparisons.

4.1. Pseudocode (Dijkstra's Algorithm)

```
for each vertex v:
    dist[v]= infinity
    prev[v]= undefined
dist[source]=0
set all vertices as unvisited
while destination is unvisited:
    v = least-value unvisited vertex
    set v as visited
    for each edge from v to neighbor w:
        if dist[v] + length(v, w) < dist[w]
            dist[w] = dist[v] + length(v, w)
            prev[w] = v
```

4.2. Flowchart





Note: To enable the reconstruction of the shortest path after the algorithm completes (as shown in the second figure), it is necessary to maintain a predecessor pointer (also called a previous-node reference) for each visited node.

This pointer should be updated whenever a shorter distance to a node is found, allowing the algorithm to trace back the optimal path from the destination node to the starting node.

5. References:

1. GeeksforGeeks. *Dijkstra's Shortest Path Algorithm (Greedy Algo-7)*, from <https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/>
2. Baeldung. *Dijkstra's Algorithm*, from <https://www.baeldung.com/cs/dijkstra>
3. Code with Red. (2020/07/05). *Depth First Search (DFS) Explained: Algorithm, Examples, and Code*. <https://youtu.be/PMMc4VslacU>
4. Reducible. (2020/09/26). *Breadth First Search (BFS): Visualized and Explained*. <https://youtu.be/xlVX7dXLS64>
5. Spanning Tree. (2020/08/15). *How Dijkstra's Algorithm Works (Pseudocode)*. https://youtu.be/EFg3u_E6eHU
6. DOUG'S WORLD. (2019/04/13). *Dijkstra's Algorithm*. <https://www.dougmahugh.com/dijkstra>