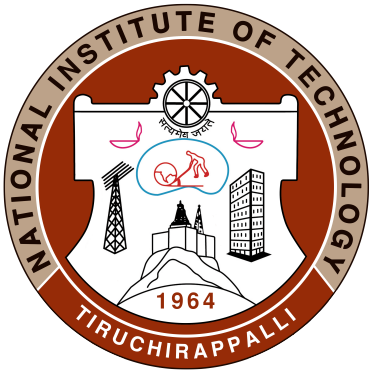
Thesis on

Route Optimizer Web Application- Snowdin



Submitted to

CoE-ERSS CDAC Thiruvananthapuram

NIT Tiruchirappalli

Submitted by:

Sidhant Varshney (**107123110**)

Vedansh Mangal (**107123132**)

Wangwad Pranav Suresh (**107123138**)

Faculty in charge

Dr. Venkata Kirthiga

Department of Electrical and Electronics

National Institute of Technology, Tiruchirappalli

May, 2025

Route Optimizer Web Application- Snowdin

Using React, Node.js and GraphHopper API

Siddhant Varshney, Pranav Suresh Wangwad, Vedansh Mangal
NIT Trichy

May 2025

Abstract

This report presents the development of a route optimization system that integrates the capabilities of modern web technologies and powerful geographic APIs. Utilizing React for the user interface, Node.js and Express.js for server-side logic, and the GraphHopper routing engine for geospatial computations, this tool allows users to calculate the most efficient routes between locations.
The system leverages the A* (A-star) search algorithm, a widely used pathfinding technique known for its balance between optimality and performance, to ensure accurate and timely route calculations. The report delves into the theoretical principles of route optimization, architecture of the application, API interactions, implementation details, and performance analysis.

1 Introduction

Efficient transportation is a cornerstone of both urban planning and logistics operations. Route optimization—the process of determining the most efficient travel path between multiple points—is critical for reducing operational costs, saving time, and minimizing environmental impact. With the rise of GPS technologies and availability of open-source geographic data, modern web applications can integrate intelligent routing functionalities.

This project explores the creation of a full-stack web application that enables users to find optimized routes. By integrating open-source APIs like GraphHopper with custom user interfaces and backend logic, this tool aims to be both accessible and scalable.

2 Theory and Background

2.1 Route Optimization Fundamentals

Route optimization is a classical problem in operational research and computer science, closely related to graph theory and combinatorial optimization. At its core, it involves finding the shortest or fastest path in a network of roads, which can be represented as a graph of nodes (intersections) and edges (roads).

Various algorithms are used for solving routing problems:

- **Dijkstra's Algorithm:** A well-known method for finding the shortest paths from a single source node to all other nodes in a graph.
- **A* Algorithm:** An extension of Dijkstra's that uses heuristics to speed up the pathfinding process.
- **Bellman-Ford Algorithm:** Useful for graphs with negative weight edges.
- **Travelling Salesman Problem (TSP):** An NP-hard problem concerned with finding the shortest route visiting multiple nodes and returning to the origin.

2.2 A* Algorithm

A* is a graph traversal and pathfinding algorithm that is used in many fields of computer science due to its completeness, optimality, and optimal efficiency. Given a weighted graph, a source node and a goal node, the algorithm finds the shortest path (with respect to the given weights) from source to goal.

We have utilized A* algorithm for the following reasons:-

- **Efficiency:** A* significantly reduces the number of nodes explored compared to uninformed algorithms like Dijkstra's, leading to faster route computation.
- **Optimality:** When using an admissible and consistent heuristic, A* guarantees the shortest possible path.
- **Heuristic Flexibility:** The algorithm allows the integration of various heuristics (e.g., Euclidean, Manhattan), enabling adaptability to different types of maps or routing scenarios.
- **Scalability:** A* performs well even on large-scale road networks, making it suitable for real-world navigation and route planning applications.

- **Integration with APIs:** A* is compatible with routing engines like GraphHopper and OpenStreetMap-based tools, making it practical for web-based geographic applications.

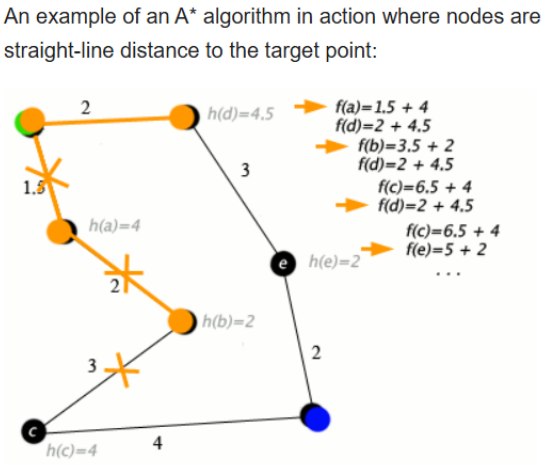


Figure 1: Example of A* Algorithm.

2.3 GraphHopper API

GraphHopper is an efficient open-source routing engine that uses OpenStreetMap (OSM) data. It provides fast pathfinding capabilities using preprocessed graph structures. The core engine supports various routing algorithms like contraction hierarchies, landmark-based routing, and flexible vehicle profiles.

Key features:

- Support for multiple transportation modes (car, bike, foot, etc.)
- Turn restrictions and road preferences
- Time- and distance-optimized routing
- RESTful API integration

2.4 Web Development Stack

React.js is a JavaScript library for building user interfaces, particularly suited for single-page applications where real-time data updates are common. It allows component-based design and state management.

Node.js is a runtime environment that allows executing JavaScript server-side. It is known for its non-blocking I/O model and event-driven architecture, making it ideal for scalable applications.

Express.js is a minimalist web framework for Node.js. It simplifies routing and middleware integration, allowing developers to set up APIs and web servers quickly.

3 System Architecture

The application consists of three main components:

1. **Frontend (React):** Handles user input, renders the map, and visualizes routes.
2. **Backend (Node.js/Express):** Processes input, communicates with the GraphHopper API, and sends results back to the frontend.
3. **Third-Party API (GraphHopper):** Performs route calculation based on geographic data.

3.1 Frontend Architecture

The frontend uses React components for modular UI construction. Address inputs use OpenStreetMap's Nominatim API for real-time search suggestions. Leaflet.js is used for rendering the map and user-selected markers.

3.2 Backend Architecture

The backend is responsible for:

- Receiving origin and destination data from the frontend.
- Formatting API requests to GraphHopper.
- Parsing and formatting the response data.
- Handling API key management and error handling.

4 Implementation Details

4.1 Frontend

The application starts with user input through address fields or map clicks. Once both points are selected, a POST request is sent to the backend. Upon receiving the response, the route is drawn using a polyline and relevant information (distance, time) is displayed.

```
const handleFindRoute = async () => {
  if (!start || !end) return;
  const response = await axios.post("http://localhost:5000/api/
     ↪ route", {
    start, end
  });
  setRoute(response.data.route.map(coord => [coord.lat, coord.lng
     ↪ ]));
  setInfo(response.data.info);
};
```

4.2 Backend

The backend constructs a request to the GraphHopper routing API using user-supplied coordinates. It handles the JSON response and forwards cleaned-up data to the client.

```
app.post("/api/route", async (req, res) => {
  const { start, end } = req.body;
  const url = "https://graphhopper.com/api/1/route";
  const ghResp = await axios.get(url, {
    params: {
      point: [`${start.lat},${start.lng}`, `${end.lat},${end.lng
         ↪ }`],
      vehicle: "car",
      locale: "en",
      points_encoded: false,
      key: graphhopperApiKey,
    }
  });
  const path = ghResp.data.paths[0];
  const route = path.points.coordinates.map(([lng, lat]) => ({
     ↪ lat, lng }));
  res.json({ route });
});
```

5 Testing and Results

Testing included:

- Inputting various origin-destination pairs.

- Verifying route rendering on the map.
- Ensuring error handling for missing inputs.
- Measuring response latency from GraphHopper.

The app consistently produced accurate routes for different transport modes. Routes matched expectations visually and the reported distance/time was realistic.

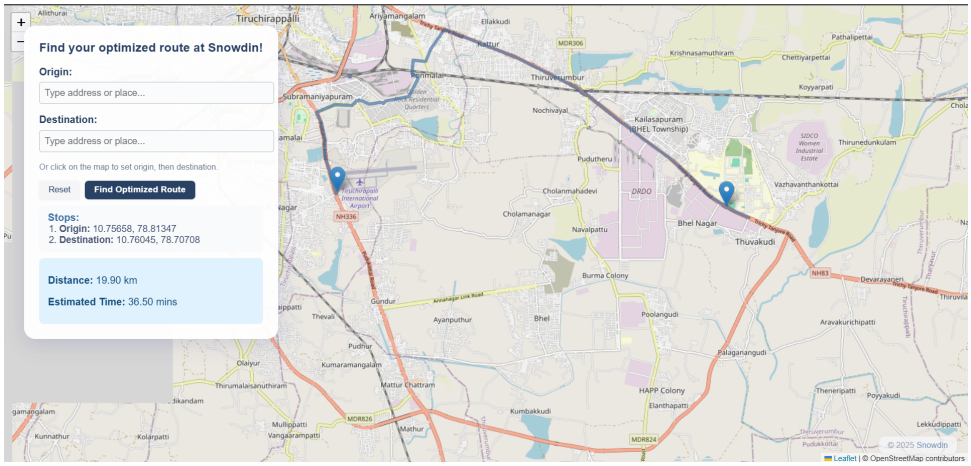


Figure 2: Distance between two points shown. (Between the two pointers)

6 Conclusion

This project showcases a practical application of full-stack web development in the geospatial domain. By leveraging React, Node.js, and GraphHopper, an efficient and interactive route optimization tool was built. The system demonstrates a scalable approach that could be extended with features like route saving, multiple waypoints, and real-time traffic data. Further improvements could include integration with additional mapping services, support for public transport modes, and performance tuning through backend optimizations.

7 References

- GraphHopper API: `https://www.graphhopper.com`
- React Documentation: `https://reactjs.org`
- Node.js Documentation: `https://nodejs.org`
- Express.js: `https://expressjs.com`

- Dijkstra's Algorithm: `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`
- A* Algorithm: `https://en.wikipedia.org/wiki/A*_search_algorithm`