TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A PROJECT REPORT ON

PATH FINDING VISUALIZER

Submitted by:
AAYUSH MISHRA(079BCT004)
ABHIYAN KHANAL(079BCT008)
ANKIT MEHTA(079BCT018)
BINI CHAND(079BCT033)

Submitted to:
Department of Electronics and Computer Engineering
IOE, Pulchowk Campus
Kathmandu, Nepal

March 10, 2025

ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to all the professors and lecturers who inspired our genuine interest in the field of Computer Science and Engineering, which has motivated us to embark on this project.

We extend our special thanks to Mrs Bibha Sthapit, for her insightful guidance, which has significantly contributed to the inception of this project.

We also sincerely thank the Department of Electronics and Computer Engineering at Pulchowk Campus for providing us with the opportunity to work on this project, allowing us to expand our knowledge and technical expertise.

ABSTRACT

Path finding algorithms play a crucial role in artificial intelligence, robotics, and computer networks by determining the shortest or most efficient route between two points. This project presents a web-based path finding visualization tool that demonstrates the behavior of four widely used algorithms: A algorithm, Greedy Best-First Search, Dijkstra's algorithm, and Bidirectional Search*. The primary objective is to provide an interactive and educational platform that improves the user's understanding of these algorithms through real-time simulation.

The project is implemented using JavaScript, React.js ensuring a dynamic and visually engaging experience. Users can interact with a grid-based map, set custom start and goal points, and observe how different algorithms explore paths in real-time. The efficiency of each algorithm is visually represented by marking the visited nodes and highlighting the shortest path found.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Path finding algorithms are computational procedures designed to find an optimal path from a starting point to a target location within a given environment or network. They typically involve traversing the graph by following edges and updating node-to-node distance estimates as new information is discovered. Some common pathfinding algorithms include Dijkstra's algorithm, the A* search algorithm, breadth-first search (BFS), depth-first search (DFS), and greedy best-first search (GBFS).

Pathfinding algorithms are commonly used in various applications such as navigation systems, routing protocols for computer networks, and game AI for determining optimal paths or strategies. They can help improve efficiency, reduce travel times, minimize energy consumption, or optimize resource allocation by finding the most suitable routes between different locations or points of interest.Different pathfinding algorithms have varying strengths and weaknesses in terms of their accuracy, computational complexity, memory requirements, and adaptability to dynamic changes or disturbances in the environment.

1.2 Objectives

The objectives of this project are as follows:

1. **Develop an Interactive Web-Based Visualization Tool:**The project aims to design and implement an intuitive graphical interface where users can visualize how pathfinding algorithms explore and determine the shortest path in a given grid. The interactive nature allows users to set their own start and goal points, and observe how different algorithms perform in real time.

2. **Compare the Performance of Multiple Algorithms:** Four major pathfinding algorithms are implemented:

   - **A Algorithm*** – Uses heuristics to optimize pathfinding.
   - **Greedy Best-First Search**– Focuses on heuristic-driven search but is not always optimal
   - **Dijkstra's Algorithm** – Guarantees the shortest path by exploring all possible routes.
   - **Bidirectional Search** – Improves efficiency by searching from both the start and goal points.

3. **Provide an Educational Tool for Learning Pathfinding Algorithms:** Many students and researchers struggle with understanding how pathfinding algorithms work, especially when studying them through static explanations in textbooks. This project aims to provide a dynamic and visual learning experience that makes algorithm behavior more understandable and engaging.

4. **Ensure an Accessible and User-Friendly Interface:** A key objective is to ensure that the user interface (UI) is intuitive, responsive, and visually clear. Users should be able to interact with the tool effortlessly, modify grid properties, and switch between algorithms without technical difficulties.

2 Theoritical Background

2.1 A* Algorithm

A (A-star)* is a very popular and efficient pathfinding algorithm used for finding the shortest path from a start node to a goal node in a weighted graph. It combines both the actual cost to reach a node and an estimate of the cost to reach the goal node, providing a balanced and optimal search.

Key Concepts

- g(n): The actual cost from the start node to node n (the distance traveled so far).
- h(n): The heuristic estimate of the cost from node n to the goal node (often calculated using a distance metric like Euclidean distance or Manhattan distance).
- f(n): The total estimated cost of the cheapest solution through node n:
  f(n)=g(n)+h(n)
  f(n)=g(n)+h(n)
- The algorithm selects the node with the smallest f(n) value to explore next, ensuring that it efficiently balances between exploring the path already traveled (g(n)) and the estimate of the remaining path (h(n)).

2.2 Dijkstra's Algorithm

Dijkstra's algorithm is a classic algorithm used for finding the shortest path from a source node to all other nodes in a weighted graph. Unlike A*, it does not use a heuristic and only considers the actual cost from the start node.

Key Concepts

- g(n): The cost to reach node n from the start node (this is the same as in A*).
- No heuristic is used in Dijkstra's algorithm, so it solely focuses on the actual cost.

2.3 Greedy Best-First Search:

Greedy Best-First Search is a pathfinding algorithm that selects the node that appears to be closest to the goal based on a heuristic function. It uses only the heuristic h(n) to guide its search, meaning it focuses on the estimated cost to the goal without considering the cost already incurred.

Key Points

- g(n): Not considered in Greedy Best-First Search.
- h(n): The heuristic estimate of the cost from node n to the goal node.
- f(n): In this case, f(n) = h(n) (since g(n) is not used).

2.4 Bidirectional Search

Bidirectional search is an algorithm that simultaneously searches from both the start and the goal node. The main advantage of using bidirectional search in pathfinding is that it can potentially reduce the search time by exploring the graph from both ends, instead of just from the start node or the goal node. The idea is that both searches will meet somewhere in the middle, thus halving the total number of nodes explored compared to a unidirectional search.

Key Points:

- It essentially performs a "forward search" from the start node and a "backward search" from the goal node.
- The goal is to meet somewhere in the middle, thus reducing the search space compared to a single-direction search.

3 Implementation

3.1 Overview

1. **Map Visualization:**
   This project is using DeckGL and MapGL to display a map and various layers, such as the pathfinding route, start/end points, and a selection radius. DeckGL is used to manage layers like PolygonLayer, ScatterplotLayer, and TripsLayer, each playing a distinct role. MapGL from react-map-gl is responsible for rendering the base map.

2. **Pathfinding:**
   The main functionality of the project revolves around pathfinding. The user can click on the map to set the start and end points, and the system will calculate and animate the shortest or optimal path based on an algorithm (like A* or Bidirectional). The algorithm's state and pathfinding process are managed with the PathfindingState model, which handles the nodes and pathfinding logic.

3. **State Management:**
   This project uses React's useState and useEffect for managing state, including the start and end nodes, trips data, current animation time, settings, and map view state. useSmoothStateChange is used to create smooth transitions for the selection radius opacity and other state changes.

4. **UI Controls:**
   The Interface component allows the user to interact with the system, including starting pathfinding, changing settings, and controlling the animation playback. The Slider component (from Material-UI) is used for user input to control various parameters like speed and radius.

3.2 System Working

1. **Map Interaction:** The user clicks on the map to select a start or end point for pathfinding. The map click handler (mapClick) checks the type of click (left or right) to determine whether the start or end point should be placed.

The getNearestNode function fetches the closest node to the clicked location, which will then be used as the start or end node for the pathfinding process. The selectionRadius state represents a circle around the start node, and the map is updated to show this radius.

2. **Pathfinding Algorithm:** The startPathfinding function initiates the pathfinding process by calling the state.current.start() method with the selected algorithm (e.g., A*).

   Pathfinding progress is animated using the animateStep and animate functions. As the pathfinding algorithm progresses, nodes are added to the waypoints array, and the path is drawn using the TripsLayer in DeckGL.

3. **UI Components:** The Interface component provides various controls like the algorithm selector, speed control (using the Slider), radius control, and play/pause buttons.

   The Slider component allows the user to adjust values for speed, radius, and other parameters. The useSmoothStateChange hook is used for smooth transitions when changing the opacity of the selection radius or other state changes.

3.3 Findings

1. Google Maps essentially uses two Graph algorithms — Dijkstra's algorithm and A* algorithm, to calculate the shortest distance from point A (Source) to point B (destination).

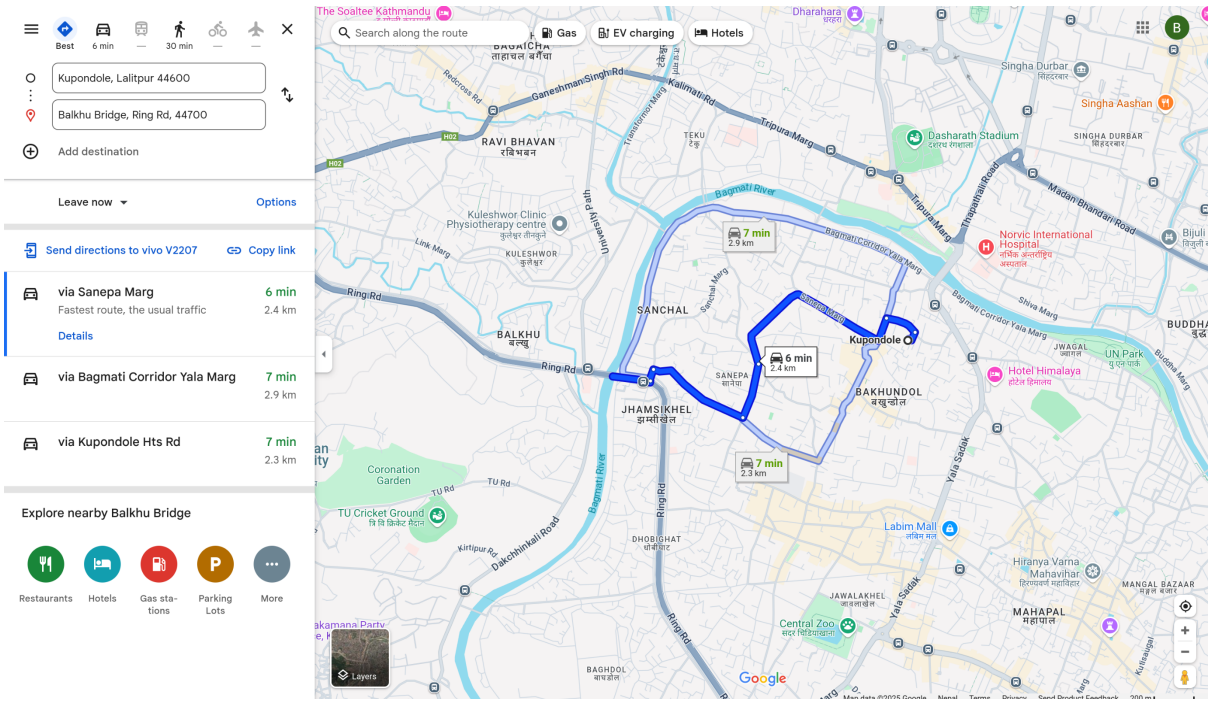

Figure 1: shortest path in Google Maps

2. Pathao,Indrive, like most ride-hailing and delivery services, likely uses the A search algorithm* for its pathfinding functionality, which is a highly efficient method for

finding the shortest route between two points on a map, taking into account factors like traffic conditions and road distances, allowing it to calculate the most optimal path for drivers to reach their destinations quickly

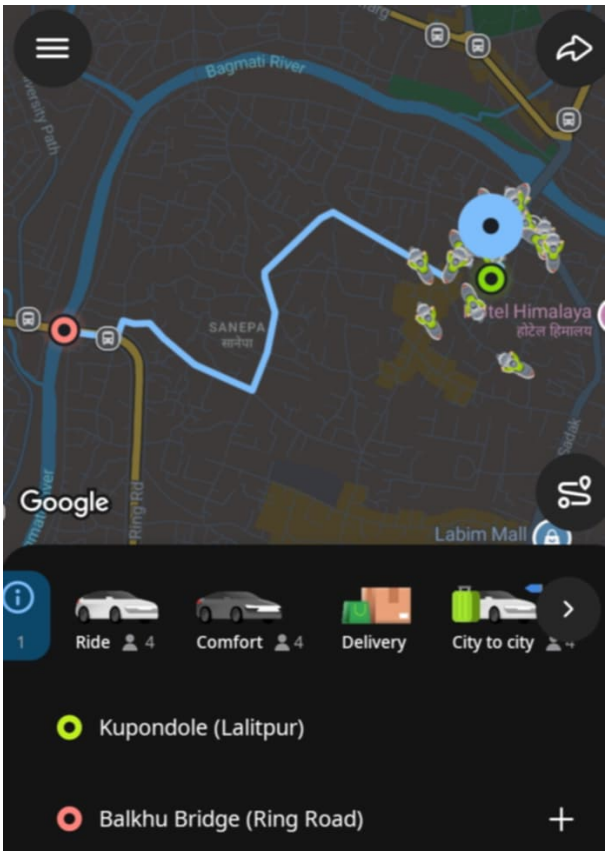

Figure 2: shortest path in Indrive ride-hailing services

4 Result

The results of this project demonstrate the successful implementation of an interactive pathfinding visualization tool using DeckGL, React, and MapLibre. This system enables users to select start and end locations on a map, choose different pathfinding algorithms, and visualize the computed paths dynamically with smooth animations.

The project tested multiple pathfinding algorithms, including A (A-Star), Dijkstra, and Bidirectional Search*, to compare their efficiency and accuracy. The following observations were made:

1. **A* Algorithm**

    - Performed efficiently by heuristically prioritizing the shortest path.
    - Delivered fast results, especially in larger maps.
    - Performance was dependent on the heuristic function used.

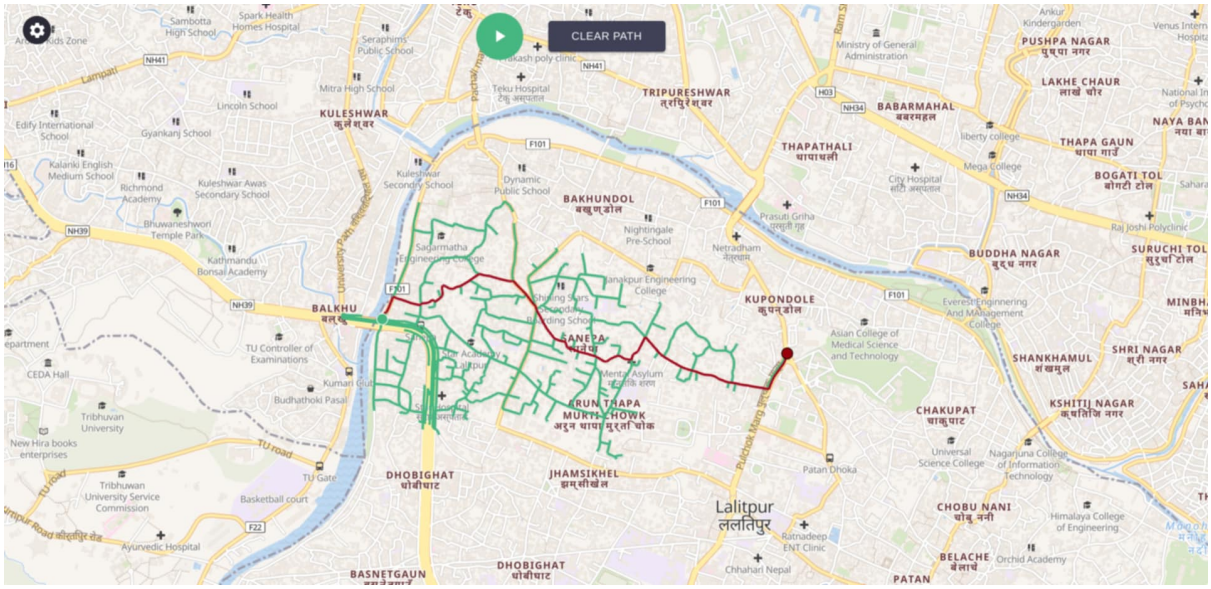

Figure 3: shortest path using A* algorithm

2. **Dijstras Algorithm**

    - Explored all possible paths, making it slower for larger maps.
    - Guaranteed the shortest path but was computationally expensive.
    - Best suited for dense networks where path cost matters more than speed.

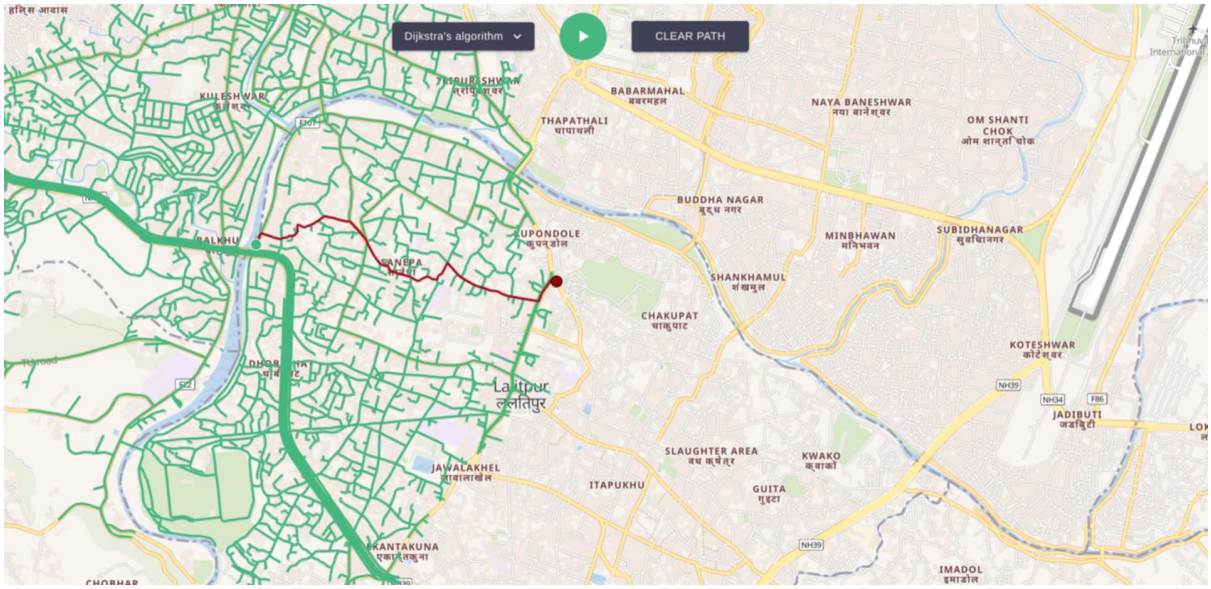

Figure 4: shortest path using Dijstras algorithm

3. **Bidirectional Search**

- Executed faster than A* and Dijkstra by searching from both start and goal nodes simultaneously.
- Reduced the number of explored nodes significantly.
- Most efficient for scenarios with clearly defined start and end points.

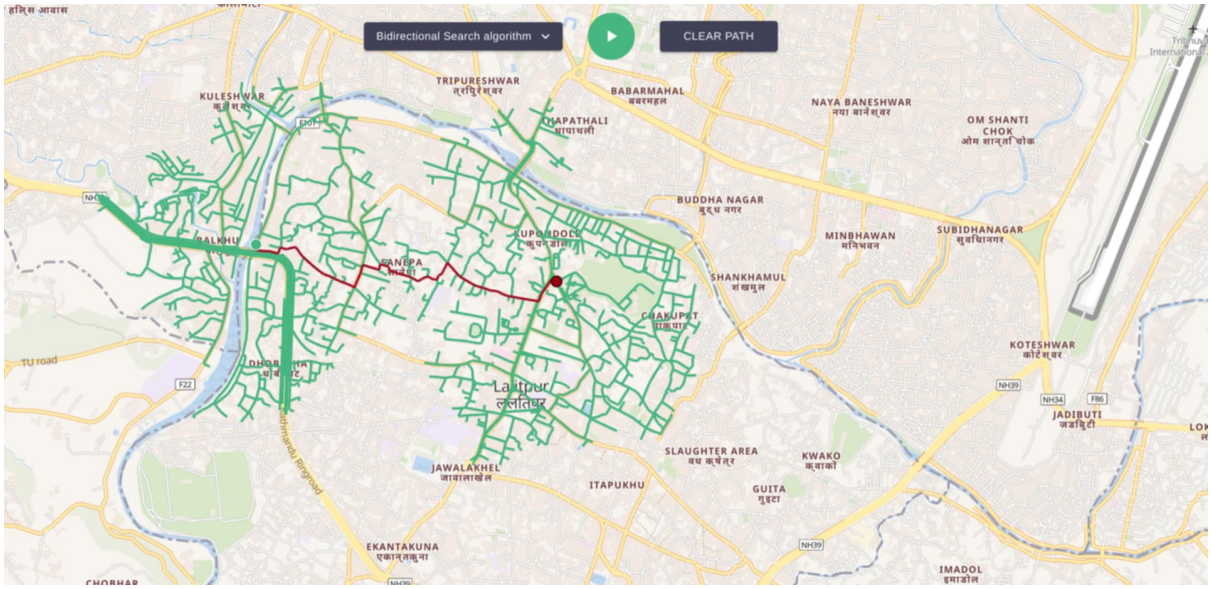

Figure 5: shortest path using Bidirectional Search algorithm

4. **Greedy Best-First Search**

   - Focuses on expanding nodes that are closest to the goal using only the heuristic function.
   - Much faster than Dijkstra but does not guarantee the shortest path.
   - In some cases, it gets stuck in suboptimal paths.

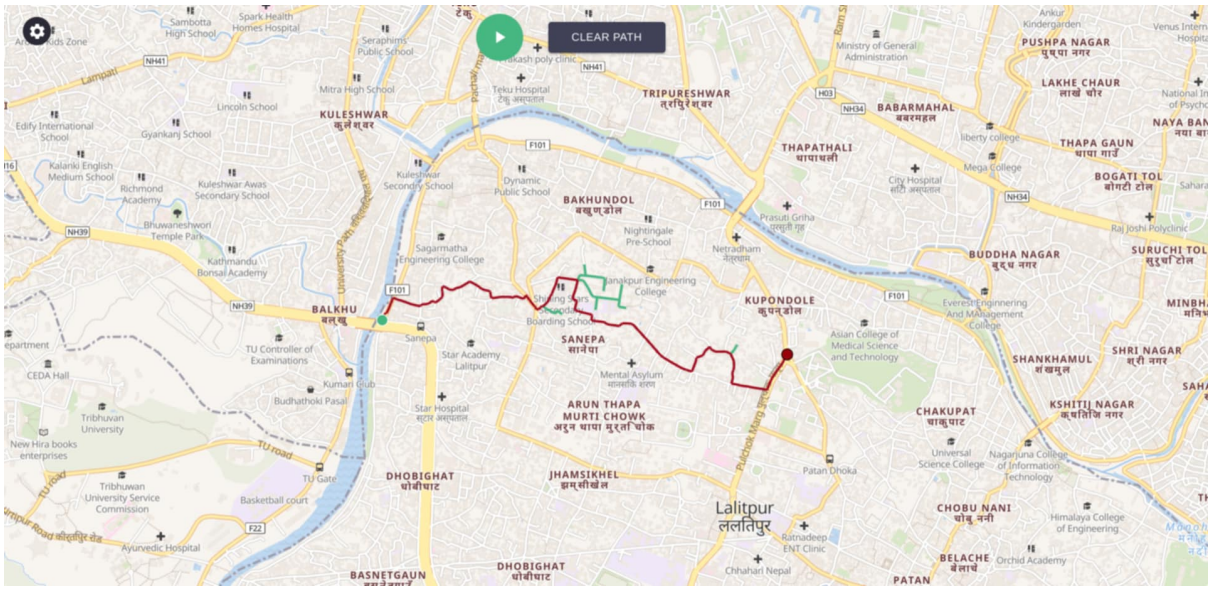

Figure 6: shortest path using Greedy algorithm

Key Observations

1. Greedy Best-First Search was the fastest, but it sometimes chose inefficient paths.
2. A* provided a balance between speed and path quality by combining heuristics with actual travel cost.
3. Dijkstra was slowest but guaranteed the absolute shortest path.
4. Bidirectional Search was overall the most efficient for well-defined start and end points.

Algorithm	**Speed**	**Accuracy**	**Best For**
A*	Fast	Best Path	Large Maps
Dijkstra	Slow	Best Path	Weighted Graphs
Greedy Best-First	Fastest	May Fail	Fast Estimations
Bidirectional	Very Fast	Good	Large Graphs

Table 1: Comparison of Pathfinding Algorithms

5 Conclusion

This project successfully implemented and analyzed multiple pathfinding algorithms, transforming raw geographic data into an interactive, visually intuitive system for route optimization. By leveraging A (A-Star), Dijkstra's Algorithm, Bidirectional Search, and Greedy Best-First Search*, we explored the trade-offs between speed and accuracy in real-world navigation scenarios.

One of the most compelling insights gained was the delicate balance between computational efficiency and path optimality. Greedy Best-First Search, while incredibly fast, often compromised on the shortest path, whereas Dijkstra's Algorithm, despite its guaranteed accuracy, lagged behind in performance. A emerged as a powerful middle ground*, offering a well-balanced approach by incorporating both heuristic estimation and cost-based decision-making. Meanwhile, Bidirectional Search proved to be the most efficient, particularly in cases where start and destination nodes were well-defined.

Beyond algorithmic performance, this project highlighted the real-world significance of pathfinding in navigation systems. Whether applied to urban traffic routing, autonomous vehicle navigation, or even game AI, the ability to dynamically compute efficient paths remains a cornerstone of modern computational intelligence. Furthermore, the integration of DeckGL and MapLibre provided a rich, interactive experience, enabling users to visually observe the step-by-step decision-making process of each algorithm.

6 References

- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://medium.com/@sachin28/the-algorithms-behind-the-working-of-google-maps-73c379bcc9b9
- https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40