Benchmarking Dijkstra's, A*, and Bellman-ford Against Google Maps with Real World Conditions

Anthony Krenek
Keegan Murphy
krene022@umn.edu
murp1846@umn.edu

Abstract

Behind the classic maps used by popular services such as Google, sophisticated search algorithms determine optimal routes for millions each day. This study aims to reflect the ability of industry-leading routing services by examining the potential of common algorithms utilizing Dijkstra, A* with heuristics, and Bellman-ford. The algorithms are further challenged by introducing changing road conditions, such as weather and closures, in the form of heuristics.
The proposed experiment benchmarks the performance of these adapted algorithms against Google Maps, the control group. Specifically, the experiment compares the relative performance, rela This research will determine the effectiveness of a lightweight, smaller-scale algorithm handling realistic routing objectives without real-time data. Using the process and results, we aim to streamline the complexities of digital routing in a contained environment, gain insight into the realistic application of common algorithms, and replicate the partial functionality of premier routing services.

1 Introduction

1.1 Problem Description

Large-scale mapping systems such as Google Maps are at the forefront of modern navigation, offering optimal routes to the public with sophisticated and efficient algorithms. However, these models are often proprietary and abstracted, with undisclosed inner workings. The layers of complexity and lack of transparency with these systems lead to many concerns. For developers, users, and the broader community, the algorithms at the forefront of our navigation systems should be transparent and thoroughly understood. In the case of severe weather, dangerous conditions, or simple road closures, it is all the more important to seek an understanding of how a routing algorithm re-routes a user. The research team intends to develop efficient routing algorithms based on pathfinding knowledge and industry practices. This further deepens the understanding of how leading routing services function through replication. This approach aims to translate complex routing algorithms into a more comprehensible format by reconstructing them within our own framework.

1.2 Objectives of the Study

- **Development of Advanced Routing Algorithms:** Design advanced routing algorithms capable of determining an optimal path between a specified origin and destination. These algorithms must dynamically adapt to changing road conditions, such as traffic congestion, road construction, and closures due to weather or emergencies, without the need for real-time data. The flexibility to customize algorithm parameters and road conditions will be necessary for this objective.
- **Performance Benchmarking:** Conduct a series of benchmark tests by comparing the proposed algorithms' route suggestions to those of Google Maps under realistic routing scenarios. The primary benchmarks will measure relative performance P_{rel} and relative efficiency E_{rel}, aiming to find an algorithm with metrics within +/- 10% of Google Maps. This will validate the algorithms's effectiveness in providing optimal routes under the same conditions.
- **Dynamic Edge Weight Adjustment with Penalties and Heuristics:** The algorithms developed will feature dynamic edge weight adjustments, utilizing penalties and heuristics to account for changing road conditions. The adaptability of these algorithms is assessed by the metric A, which the research group aims to optimize to 100%. This metric demonstrates the algorithms' ability to adjust routing strategies in response to environmental changes dynamically. The absence of a control group is noted, and the research group will empirically review the adjustments.

1.3 Significance of the Problem

Developing, understanding, and testing common pathfinding algorithms gives insight into their potential use cases and effectiveness. Reliable and efficient algorithms lead to equally effective routing systems, which are relied on by officials, businesses, and the public in general. The proposed algorithms are used to replicate and understand the functionality of these industry-leading providers on a smaller scale, aiming to provide insights into the algorithms behind pathfinding. The information gathered is represented through the proposed algorithms of this report, which comprehensively aims to route realistic scenarios with a minimalist framework. Although it is not our goal to replace or undermine premier routing systems such as Google Maps, the research in this report offers insight into the common algorithms and theories utilized in routing.

1.4 Organization of the Report

This report is divided into five main sections to suit the process of experimenting with the proposed algorithms. Section 2, Background, focuses on existing theory and methodology on various algorithms, heuristics, and industry services. This routing research forms the foundation of the knowledge used throughout this report. Section 3, Routing Model, covers the proposed solution's design, approach, and implementation. This routing model is benchmarked through an experiment described in Section 4, and the experimental results are further reported and analyzed in Section 5, Results. A final analysis of the Results section is in Section 6, the report's Conclusion.

2 Background

Extensive theoretical and practical research has been done in routing, mapping, and pathfinding. These fields leverage various algorithms and heuristics to efficiently navigate and optimize paths in complex environments. This section reviews foundational theories, key algorithms, and the diverse heuristics that enhance pathfinding in robotics, mapping, and other software applications. The information is then applied to routing to determine effective ways to find the shortest route while accounting for changing road conditions. To gauge the effectiveness of the proposed model, existing services in the industry are experimentally analyzed and studied, providing insights into how they manage complexities and variability in routing challenges. This comparative assessment helps identify areas where the proposed model excels or requires further refinement.

2.0.1 Algorithms and Approaches to Routing Problems. This background research group highlights the algorithmic and practical approaches to solving routing problems. Then, the information is further specialized and broken down to fit the problem statement. Central to these processes is the A* algorithm, renowned for its effectiveness and simplicity in finding the shortest path between points. Further, A* has the powerful ability to optimize its search with Heuristic functions. These articles, papers, and conference papers give a strong theoretical approach to routing, pathfinding, and properly working with multiple diverse algorithms and heuristics.

2.0.2 Comparative Analysis between Dijkstra and Bellman-Ford Algorithms in Shortest Path Optimization. The group's goal is to be able to test the routes that different shortest path algorithms create and to see how efficiently they are able to create them. This study shows an analysis between Dijkstra and Bellman-Ford, two algorithms that will be used in the program and analysis later on. This analysis makes different modifications to the two algorithms to see how they can perform with n nodes. Dinitz and Itzhak presented a new hybrid algorithm called BFD the Bellman-Ford-Dijkstra [1]. They created this by combining Bellman-Ford and Dijkstra, and their ability to locate the shortest paths has been tested. Further, the runtime was improved by adding shared destructive cost edges.
Simulation Results: The simulation was conducted on a different number of nodes. The number of nodes they used was five, 10, and 50. These results also only use Dijkstra and Bellman-Ford for the algorithms. For the timing results, it is clear that with an increasing number of nodes the runtime increases for Bellman-Ford. For example, going from five to 10 nodes, the runtime increased by 243 milliseconds [1]. When going from 10 nodes to 50, the time also increases dramatically. One thing that was very shocking from the results was Dijkstra's. The results show that the faster the running time is with an increased number of nodes. For example, when using 10 nodes the run time is 3724 which is still slower than the Bellman-Ford algorithm [1]. When 50 nodes are present, the runtime of the algorithm was 2072 milliseconds [1]. Comparatively, using five nodes with Dijkstra's gave a runtime result of 1351 milliseconds [1]. This result was very shocking, as Dijkstra yielded a slower runtime with 10 nodes than 50. From these tests and analysis, it is clear to see that Dijkstra was faster when using big and small graphs to get the shortest path.

2.0.3 Engineering Route Planning Algorithms. This paper gives an overview of the advancements in shortest-path algorithms for computing shortest paths in road networks. The main focus of this paper is static road networks, including fixed edge costs, travel time, distance, and other factors [14]. The innovations in static routing in shortest path algorithms, such as A* search and Dijkstra's, are crucial to the result. Evolution in these algorithms is related to Highway Hierarchies, Contraction Hierarchies, and Transit-Node routing. The scope of using these evolutions aims to balance query speed, space efficiency, and processing time. The paper also outlines the difficulties of different scenarios where edge weights change with time. Finding shortest paths through road networks is time dependent and there are other problems we have to account for. This presents the challenge of correctly identifying how these problems can be represented in an algorithm. By implementing these algorithm changes, we can scale solutions on huge road networks [14].

2.0.4 A fast, efficient technique for finding a path through multiple destinations. The proposed toolset Siddharth uses for effectively calculating a route is composed of pathfinding algorithms, heuristics, and Google Maps. This comprehensive report highlights a variety of important routing components, most importantly being the use of Dijkstra's and A* algorithms [3]. These two algorithms can be properly

adjusted with heuristics to effectively identify short paths [3]. Siddharth additionally emphasizes the importance of time and space complexity to ensure the routes are calculated in a reasonable amount of time. This contributes to the pathfinding field by giving a minimalist approach to calculating routes, where some mass-scale, distributed systems require a hierarchical routing algorithm.
This report is entirely applicable because it demonstrates a potential approach to a similar question. Using similar strategies of pathfinding a grid with edge weights and heuristics, the algorithm proposed can effectively route paths with a bloat-free implementation. The calculations can be benchmarked against Google Maps API in a similar fashion, potentially finding some alternate ways to efficiently route paths.

2.0.5 Application and Improvement of Heuristics in A*. This conference paper describes the advanced applications of heuristics, including selecting a an optimal heuristic to maximize efficiency. This is applicable to the report as using a heuristic that is the most applicable for the situation will significantly improve the pathfinding efficiency. The conference paper does this by comparing four common heuristics: Manhattan distance, Euclidean distance, Chebyshev distance, and Diagonal distance [7]. Further, the four heuristics are tested in an experimental setting on a 2d grid map, which is similar to the grid setup used for the proposed model [7]. For the proposed routing model which is being developed, Chebyshev distance can be applied to A* to significantly improve the performance and will be used within the model.

2.1 Heuristics for Real Time Data and Changing Road Conditions

This subsection highlights research tailored towards realistic routing scenarios that can occur during a simulation. These scenarios include many changing road conditions, such as weather, traffic, and road closures. These are implemented through heuristics and changing edge weights, which slightly alter the optimal route to navigate around them. Adding changing road conditions allows the proposed model to simulate realistic routing scenarios. The following papers, articles, and conference papers give background information and strategies to implement this powerful feature into the model.

2.1.1 Traffic congestion prediction based on Estimated Time of Arrival. Although not directly tied to pathfinding algorithms, this article describes some of the aspects of routing procedures. For example, real-time data aggregation and traffic congestion are referenced. These factors affect the ETA and heuristics of a route, both while computing the route and actually navigating it [16]. This contributes to the report's analysis by providing additional information on exactly how heuristics affect route calculations and what those heuristics are. That assists greatly with the contextual analysis of routing algorithms and can be paired with the information in the source "Vehicle Route Planning using Dynamically Weighted Dijkstra's Algorithm with Traffic Prediction" [15]. Regarding the proposed solution's software component, this article also provides software pieces and diagrams to explain how real-time information affects the route [16]. There are also significant results when the information is applied to existing routes through Google Maps API, especially in the context of ETA and traffic.

2.1.2 Real-time routing with OpenStreetMap data. OpenStreetMap is an open-source method of obtaining route information to properly calculate an efficient, safe, and realistic path. This includes real-time data which is used as a heuristic for on-the-fly adjustments to a route. Even though it is open source, the results of this conference paper prove the reliability of the service. This service can provide road data and real-time data, which is useful for the coding section of the proposed algorithm [11]. Combined with Google Maps API, there can be multiple benchmarks and comparisons of the proposed algorithm across mapping services. There are additional sections on Dijkstra's and A* algorithms and their use with OSM services, which contribute to the analysis presented in "Engineering Route Planning Algorithms" and ensure a proper approach to the questions asked [14]. The additional sections of information in the conference paper provide more context for route planning, real-time information, and algorithm implementation [11]. Since the Open Street Map service is open source, there is more context on the engineering for the back-end servers. Google's Mapping service is kept under a proprietary lock, so the information for analysis there is much more stringent. This context is good for the analysis and research necessary for the proposed solution.

2.1.3 Vehicle Route Planning using Dynamically Weighted Dijkstra's Algorithm with Traffic Prediction. One roadblock when using shortest path algorithms is determining the edge weights to accurately represent different traffic conditions. A study conducted at Cornell University introduced a "Modified Dynamic Weight Dijkstra's Algorithm" that utilizes traffic flow theory [15]. This theory considers the average number of passing vehicles, the number of vehicles occupying a space, and the average speed, represented by the equation $q(i,j) = k(i,j) * u(i,j)$ [15]. This approach transforms edge weights into dynamic values that change over time, representing travel times more accurately. The accuracy of this model in reflecting traffic networks is crucial. When there is a change in edge weights at different times, the algorithm reroutes and updates the travel times to the affected nodes [15]. To reconstruct the shortest path from the source to the destination, a parent array is utilized. This array is promptly updated whenever a quicker route is identified

[15]. The implementation of this algorithm will be highly beneficial for comparisons between different algorithms and the time it takes to find the most optimal routes.
Results: This method was tested on the road networks of five major cities around the globe, represented as a graph [15]. The conventional Dijkstra's algorithm, which does not utilize traffic flow theory, yielded longer travel times in units of time. The shortest path found by the standard algorithm took 46 units of time [15]. In contrast, the Modified Dynamic Weight Dijkstra's Algorithm found the shortest path in just 36 units of time, reducing travel time by 21.7% [15]. By incorporating traffic flow theory, this will anticipate traffic movements and adjust the edge weights accordingly, allowing insight to find and predict the most optimal routes.

2.2 Map Services and Resources

This final background suubsection highlights the studies and information surrounding existing mapping services. One such mapping service, Google Maps, will be used in the experiment to establish a control group. Google Maps is well-recognized for its simplicity, public access, and powerful capability, which our model intends to replicate on a smaller scale. The following material in this section describes many routing services and how their impressive routing functionality and technology can be utilized in the experiment and report.

2.2.1 A Broad Analysis of Map Routing Applications. This conference paper has a significant amount of simplified information on mapping and routing services. This includes information on leading providers, traffic information, GPS data, platforms, usage surveys, and more [13]. Additionally, there is quantitative data that shows the disparity in calculations across services. Even though all the leading providers, such as Google, Bing, and Apple, want the same shortest route, results show the algorithms used to calculate this route differ significantly and lead to different times completely [13].
Although this information provides little technical insight into the inner mechanics of routing algorithms, this conference paper provides a zoomed-out introduction to the topic of mapping and routing. This piece can fit in as context for the introduction of the analysis performed for mapping services and has substantial detail for introducing readers. When paired with "Google Maps", it can substantiate information researched on Google Maps and the Google Maps API, which is the leading service analyzed in the proposed solution[12].

2.2.2 Comparing Alternative Route Planning Techniques: A Comparative User Study on Melbourne, Dhaka, and Copenhagen Road Networks. This user study compares the quality of alternative routes generated by the techniques of Google Maps, Plateaus, Penalty, and Dissimilarity. The penalty is a technique to compute shortest paths, but it applies a heavier weight when the shortest path is found [9]. Plateaus were created by first generating two shortest paths. One is a forward shortest path, and the other is a backward shortest path [9]. Those two trees are joined together. Generating alternative paths with plateaus is locally optimal. Dissimilarity is the technique of adding to the results set "P." The aim is to actively add the paths in an ascending order [9]. The only constraint is that the shortest paths need to be different in order for dissimilarity to work. Examining the results of using these different techniques showed that dissimilarity was the least similar. Due to the way it is set up, if paths are very similar, dissimilarity will not work. Google Maps suggested the best alternative routes, but they also had the most data, such as real-time traffic data or historical data for query issues. All three techniques (Penalty, Plateaus, and Dissimilarity) produce alternative routes.

2.2.3 Google Maps. Google is one of the leading map services in the world, providing millions of people and businesses with high-quality routing services for decades. This article provides a wide array of background knowledge on Google Maps, including history, how it functions, and the technical components that allow it to function at scale [12]. Since the proposed solution includes an analysis of existing resources, this article is incredibly useful for how Google runs its Map services. It can provide additional insights into the inner mechanics of the Google Maps API, which, when used in conjunction with the "Google Maps Platform Documentation," can provide extensive information on how one of the largest routing services operates ; further, it provides information on how the API could be utilized in other routing applications which the group plans to build[4].

2.2.4 Dijkstra's Algorithm and Google Maps. This article outlines the importance of using Dijkstra's algorithm in terms of the evolution of Google Maps and route planning. Dijkstra's was the beginning of Google Maps in terms of any navigation system [8]. The implementation of Dijkstra's on a small scale was each intersection was considered by a node and the street as an arc [8]. On a larger scale, cities were considered nodes, and interstates were considered arcs. The article also states that using Dijkstra's to compute the shortest path in routes was evolutionary to the modern technology of navigation [8]. Google Maps continues to innovate its navigation systems to avoid delays and traffic. If it was not for Dijkstra, navigating technology in general would not be as good as it is today.

2.2.5 Google Maps Platform Documentation. Built on the backbone of Google LLC, the Google Maps API is widely adopted and public, showcasing the capabilities of refined pathfinding strategies applied on a massive scale. This additionally contributes towards the field of pathfinding by providing a widely available public service that is highly optimized – all under a simplified API and UI [4].

This API provides programmatic functionality that is useful for benchmarking the performance of any proposed solutions [4]. If a calculated route is slower than what Google Maps API determines, the proposed algorithms are under performing. On the other hand, a proposed algorithm that performs better than a route generated by the Google Maps API shows that the proposed algorithms are more efficient than average (to a certain extent). In general, the Google Maps API can be used to establish a control group for the experiment. Comparing the experimental model to the control group's results can give an overall idea of how the algorithm performs comprehensively.

3 Optimizing Algorithms

3.1 Implementation

The software component, including the model and benchmarking suite, is entirely composed of Python. This choice was primarily made because Python has a wide selection of well-documented libraries. It is also well-recognized for its simplicity, leading to a streamlined development experience. The code is run on the Google Colab service, ensuring consistent computing as well as low latency in the case of network requests.

3.2 Chebyshev Distance

The Chebyshev distance heuristic calculates the maximum of the absolute differences between the coordinates of two nodes. This measure is particularly appropriate for use with the A* algorithm in grid-based pathfinding scenarios, as it effectively handles diagonal movement.Chebyshev distance treats all eight possible directions of movement (vertical, horizontal, and diagonal) as having equal cost. This characteristic makes it highly suitable for applications where movement is not restricted to just the four cardinal directions, enabling more efficient and direct pathfinding.

```
Function ChebyshevDistance(StartNode, EndNode, Graph):
StartNode_X = Graph.Nodes[StartNode]['x']
StartNode_Y = Graph.Nodes[StartNode]['y']
EndNode_X = Graph.Nodes[EndNode]['x']
EndNode_Y = Graph.Nodes[EndNode]['y']
Diff_X = Absolute(StartNode_X - EndNode_X)
Diff_Y = Absolute(StartNode_Y - EndNode_Y)
ChebyshevDistance = Maximum(Diff_X,Diff_Y)
Return ChebyshevDistance
```

3.3 Rush Hour Condition

Incorporating time-dependent weights into pathfinding algorithms enhances route planning realism by accounting for variable traffic conditions at different times of the day. By increasing weights on road segments during designated rush hours, the algorithm more accurately simulates the impact of rush hour congestion. This is important as traffic flow fluctuates throughout the day, significantly affecting ETA. The enhanced weights are set higher during peak traffic times to reflect the slower traffic movement, thus directly influencing the algorithm's route selection to avoid heavily congested areas during these periods. In this simulation, the hours defining rush and non-rush periods were chosen arbitrarily to illustrate how the algorithm can adapt to real-time traffic data similar to systems like Google Maps, dynamically altering route suggestions based on prevailing traffic conditions.

```
Function Rush Hour Weights
    If 7 <= hour < 8 then Return 3.0
    Elif 8 <= hour < 9 then Return 4.5
    Elif 9 <= hour < 10 then Return 3.5
    Elif 10 <= hour < 16 then Return 2.5
    Elif 16 <= hour < 18 then Return 4.5
    Elif 18 <= hour < 19 then Return 3.0
    Else Return 1.5
```

3.4 Road Type Weights:

In route planning, considering the speed limits and road types is crucial for accurate ETA calculations. Different road types typically have varying speed limits, which significantly affect travel time. For example, motorways often allow speeds up to 70 MPH, resulting in faster travel over longer distances, whereas residential streets typically have speed limits around 25 MPH, reflecting their more congested and pedestrian-oriented nature. By assigning different weights to these road types in our algorithm, we adapt the route cost calculations to better reflect these different road types in potential travel speed. This method attempts to illustrate the realism of our route planning, similar to how Google Maps offers users multiple route options with unique ETAs, each calculated based on the specific roads involved in the route. This approach not only optimizes for shorter ETAs but also aligns with how drivers select routes in real-world conditions.

```
Function AddRoadTypeWeights(RoadGraph):
    For each edge (u, v, d) in RoadGraph:
      road_type = d.Get('highway', 'unclassified')
        If road_type == 'motorway' then:
            d['type_weight'] = 1.0
        Elif road_type == 'trunk' then:
            d['type_weight'] = 1.2
        Elif If road_type == 'primary' then:
            d['type_weight'] = 1.5
        Elif If road_type == 'secondary' then:
            d['type_weight'] = 2.0
        Elif If road_type == 'tertiary' then:
            d['type_weight'] = 2.2
        Elif If road_type == 'unclassified' then:
            d['type_weight'] = 2.5
        Elif road_type == 'residential' then:
            d['type_weight'] = 3.0
        Else:
            d['type_weight'] = 1.5
```

3.5 Severe Weather Condition

Weather conditions play a crucial role in driving, as they can significantly impact road conditions and driver behavior, ultimately affecting the ETA for a route. Severe weather conditions such as heavy snowfall, heavy rain, and icy roads can lead to reduced visibility, slippery surfaces, and slower average speeds, resulting in longer travel times. Considering these factors when calculating route weights allows for more accurate and realistic route planning, as routes may need to be adjusted based on current weather conditions to account for potential delays.

```
Function CalculateWeatherImpactFactor(weather):
    If weather is 'clear' then:
        Return 1.0
    Elif weather is 'fog' then:
        Return 2.0
    Elif weather is 'rain' then:
        Return 2.3
    Elif weather is 'icy' then:
        Return 2.8
    Elif weather is 'snow' then:
        Return 3.0
    Elif weather is 'heavy rain' then:
        Return 3.2
    Elif weather is 'heavy snow' then:
        Return 3.5
    Else:
        Return 1.0
```

4 Experiment

The experiment design will determine how the model proposed in this report compares to existing routing tools in various realistic scenarios. The subsection *Benchmark Design* describes the layout of the benchmark suite and the services used as the control group for expected performance. The subsection *Benchmark Metrics* describes the results that are evaluated in greater detail. This clarifies the objectives of each test and how these values are used to gauge performance between the proposed routing model and established models. The Experiment section will finally conclude with *Test Scenarios*, describing the specific simulations to which each model will be exposed. This includes the routes that are planned, changing road conditions that are applied, and how each of the models is utilized.

4.1 Benchmark Design

The experiment will be performed by running the proposed model through various scenarios. This process simulates realistic model usage and tracks the results and response to the simulation. Each situation comprises a new route, algorithm, and changing road conditions, isolating a new variable each time. The same situation is executed against the control model as well. The three metrics determining the proposed model's effectiveness are relative performance, relative efficiency, and model adaptability. The relative performance is results-based and solely focuses on the model's capability to route optimally. The testing script additionally tracks the execution time of each model and uses the time to determine a relative efficiency ratio. The final experimental metric, model adaptability, focuses on the model's ability to adjust the routing results and propose alternative routes for changing road conditions. The model adaptability is empirically evaluated, checking that the altered route is correct, safe, and optimal in facing challenges. When benchmarking the proposed model, a sequence of Python scripts simulates the predefined scenarios. Third-party APIs, such as the Google Maps API, make the approach more consistent and automated, specifically allowing integration with the Python library `googlemaps`.

```
Load 'testing_locations.csv'
Load 'algorithms_output_log.csv'.
Strip and clean column names.
Convert Total Times to numeric values.

Initialize Google Maps Client with API key.
Define origin and baseline time.

Function google_api(origin, destination, api_key):
    Start timer.
    Fetch directions using Google Maps API.
    Calculate request time.

    If directions found:
        Convert trip time to minutes.
    Return trip time and request time.

For each route in algorithms' output log:
    Fetch destination coordinates.
    Calculate route from origin to destination.
    Output Google trip time and request time.

    Prel =
        (Dijkstra Time / Google Time) * 100
    Erel =
      (Google Request Time / Dijkstra Runtime) * 100
    Prel = min(Prel, 200)
    Erel = min(Erel, 200)
    Log Route, Prel, and Erel.

Output benchmark log
End For
```

4.2 Benchmark Metrics

The metrics will determine the proposed model's relative performance, relative efficiency, and overall adaptability compared to the control group during the experiment. These metrics measure various factors against the control group and check the proposed model's capability in realistic scenarios.

4.2.1 Relative Performance (P$_{rel}$). The primary factor, relative performance, will be calculated by comparing the route trip time each model calculates. Although all models slightly differ in the way they find the shortest route, the estimated trip times should be marginally close. If the proposed model has a route trip time higher than the control group, it is not performing relatively as well. Let P_{rel} be the Relative Performance. It can be defined as:

$$P_{\text{rel}} = \left(\frac{TT_{\text{control}}}{TT_{\text{experimental}}}\right) \times 100\%, \quad \text{where } P_{\text{rel}} \leq 200 \quad (1)$$

where $TT_{experimental}$ is the route trip time measured by a proposed algorithm, and $TT_{control}$ is the route trip time measured by the control group for comparison. An P_{rel} value of 100% indicates the relative performance of the proposed algorithm is identical to the control group. If P_{rel} is more than 100%, the proposed algorithm found a faster route than the control group found. However, this could be because real-time data is not incorporated into the experimental algorithm. An P_{rel} under 100% would otherwise indicate that the proposed algorithm finds less optimal routes than the control group. These calculations are made with the assumption the map data gathered from [place] is entirely accurate.

4.2.2 Relative Efficiency (E$_{rel}$). The relative efficiency of the proposed algorithm is measured to determine the overall performance and complexity when finding the shortest route. This benchmark metric is based on how fast the pathfinding algorithm is executed for a specific scenario. The Efficiency (E_{rel}) of the proposed routing algorithm can be given by the formula:

$$E_{\text{rel}} = \left(\frac{AR_{\text{control}}}{AR_{\text{experimental}}}\right) \times 100\%, \quad \text{where } E_{\text{rel}} \leq 200 \quad (2)$$

where $AR_{experimental}$ is the algorithm runtime, the time taken by the algorithm to compute the route, and $AR_{control}$ is the computation time measured from the control model. An E_{rel} value of exactly one shows the computation time of the proposed algorithm is identical to the control group. The algorithm is less relatively efficient than the control group if the E_{rel} is less than 100%. An E_{rel} value of more than 100% shows the algorithm is relatively more efficient than the Google Maps algorithms. The E_{rel} value does consider network API request time, which affects the potential efficiency of all control routing providers and their respective routing algorithms.

4.2.3 Overall Adaptability (A). The adaptability value assesses the proposed algorithm's ability to adapt to changing road conditions. This adaptability highlights the algorithm's ability to handle routing under realistic challenges. In the case of severe weather, closures, or traffic jams, the optimal route may be impeded, which leads to significant delays and safety concerns for the trip. The algorithm will modify the route slightly using a heuristic to navigate these situations. The Adaptability (A) score of the proposed routing algorithm is calculated with the formula:

$$A = \left(\frac{\text{\# of effective adaptations}}{\text{Total \# of proposed adaptation scenarios}}\right) \times 100\% \quad (3)$$

$$A_{\text{Pass/Fail}} = \begin{cases} \text{Pass}, & \text{if } A \geq 80\% \\ \text{Fail}, & \text{if } A < 80\% \end{cases} \quad (4)$$

A control group is not included in the calculation of the A value to concentrate on the proposed model's intrinsic heuristic ability that was developed rather than a comparative analysis of existing services. This approach is also necessary, as existing services rely on their own unique suite of algorithms and real-time information that often do not allow the introduction of user-defined, changing road conditions. The empirical comparison to determine if a route is effectively adapted is if the estimated trip time, algorithm runtime, and planned route are realistic for the new condition. The comparison will also occur between the algorithm with the added condition and the base performance algorithm. If 80%+ of the routes are considered to meet the conditions (M.C.), then A is a pass, A_{Pass}. Otherwise A will be considered failing adaptability requirements, A_{Fail}.

4.3 Test Scenarios

The scenarios that will be applied to the testing suite are as follows:

1. Origin Coordinates for all routes is the Alpha Kappa Psi House in Dinkytown Coordinates: Latitude: 44.98255 Longitude: -93.23867
2. Route 1 is a McDonald's in Roseville, Minnesota Coordinates: Latitude: 44.730289 Longitude: -93.475021
3. Route 2 is Mystic Lake Casino Coordinates: Latitude: 45.00362 Longitude: -93.167557
4. Route 3 is Mall of America Coordinates: Latitude: 44.856529 Longitude: -93.239449
5. Route 4 is Duluth, Minnesota Coordinates: Latitude: 46.01885 Longitude: -95.32675
6. Route 5 is Lake Miltona, Minnesota Coordinates: Latitude: 46.786671 Longitude: -92.100487

5 Results

Each of these three algorithms listed below was used in our computations to analyze how custom weight conditions on different algorithms affect the route taken to get from an origin to a destination. These changing road conditions significantly altered the algorithms' calculations. Dijkstra and Bellman-Ford were tested with each of the three conditions: rush hour, road type, and severe weather conditions. The results for each test were compared against each other to determine which algorithm was the most efficient in finding shorter routes and compared against Google Maps API.

5.1 Google Maps API Results

Google Maps API was the foundational benchmark for evaluating various shortest-path algorithms incorporating custom weights. This choice was strategic, ensuring that the paths computed were accurate and provided a reliable comparison standard Table 1. Given that Google Maps is a leading provider of routing services, it naturally forms a robust baseline for such experimental studies. Moreover, using Google Maps as a time metric was crucial in comparing different algorithms' efficiency, respective heuristics, and the impact of varying edge weight conditions. By observing the execution times recorded by Google Maps for computing routes, researchers could gain valuable insights into the expected duration for route calculations, setting a practical standard for algorithm performance in real-world applications.

Table 1. Performance of Google Maps API

Route	Est. Trip Time	API Request Time
Route 1	9 mins	0.10 sec
Route 2	35 mins	0.07sec
Route 3	16 mins	0.11 sec
Route 4	135 mins	0.11 sec
Route 5	135 mins	0.12 sec

5.2 Dijkstra's Results

The results of our experimentation with different weight conditions in route computation reveal many different variations in the ETA for each destination. Comparing the outcomes obtained using normal Dijkstra's algorithm with those derived from our modified algorithms, which incorporate various weight factors, provides valuable insights into the impact of different conditions on route planning.

Under no weight conditions, Dijkstra produced consistent ETA's for the routes computed Table2. However, when real-world weight conditions were introduced, the results showed multiple different changes in the ETA time computed Table. For example, when a road type condition weight was introduced, prioritizing faster road types like highways, the results showcased a faster route and deductions in travel time for most routes. The increase in average speed limits led to more optimal shorter routes, which resulted in shorter ETA times.

Similarly, during rush hour periods characterized by peak hour traffic, the ETA's were further reduced from the base case Table 3. This is in favor of this, too, and shows that routes with a higher speed limit average allow for faster routes.

Comparatively, when severe weather conditions were simulated, the trip times increased for all routes compared to other weight conditions Table 5. This showed the great impact of adverse weather on travel times and highlights the importance of considering such factors in route planning.

We observed similarities and differences when comparing the results to those obtained from the Google Maps API. While Dijkstra's algorithm often computed shorter total times for most routes compared to Google Maps, it's essential to note that the routes generated for comparison with Google Maps did not incorporate any publicly known weight conditions. Additionally, the Google Maps API demonstrated significantly quicker route generation times compared to our computations using Dijkstra's algorithm. This disparity in execution time suggests that the routes generated by the Google Maps API may be more efficient, likely due to the comprehensive data and algorithms utilized by the API.

Table 2. Base Performance of Dijkstra's Algorithm

Route	Est. Trip Time	Algorithm Runtime
Route 1	10.97 min	14.3 sec
Route 2	38.03 min	31.76 sec
Route 3	19.36 min	15.41 sec
Route 4	131.25 min	55.88 sec
Route 5	127.28 min	69.43 sec

Table 3. Dijkstra's Algorithm Adaptability under Rush Hour Conditions

Route	Est. Trip Time	Algorithm Runtime
Route 1	6 min	13.32 sec
Route 2	30 min	31.34 sec
Route 3	16 min	14.52 sec
Route 4	141 min	56.09 sec
Route 5	126 min	69.03 sec

The Table 6 showcases the comparative performance of Dijkstra's algorithm against Google Maps API, as depicted in the tables. Dijkstra's algorithm focuses similarly on relative performance P_{rel} and relative efficiency E_{rel}. The metrics demonstrate that Dijkstra's algorithm offers competitive performance compared to Google Maps. Specifically, the P_{rel} values for Dijkstra's algorithm range from 82% to 106%, with an average closely mirroring that of Google Maps, indicating that while Dijkstra's algorithm sometimes finds routes

Table 4. Dijkstra's Algorithm Adaptability under Road Type Penalty

Route	Est. Trip Time	Algorithm Runtime
Route 1	6.3 min	14.24 sec
Route 2	29.19 min	31.54 sec
Route 3	15.47 min	13.53 sec
Route 4	132.45 min	55.30 sec
Route 5	127.81 min	67.11 sec

with similar or slightly longer trip times, it remains quite effective. In terms of efficiency, Dijkstra's algorithm suffers by a significant margin, spanning 0%-1%, albeit without the sophisticated adaptability to real-time data that Google Maps possesses. Dijkstra's algorithm additionally has strong adaptability A, shown in Table 7. The adaptability tests Meet Criteria across the board and additionally Pass for an average. An A_{Pass} conclusion shows Dijkstra's ability to produce accurate, plausible, and realistic results when facing challenges. Those challenges were severe weather, road-type penalty, and traffic jams/high traffic.

Table 5. Dijkstra's Algorithm Adaptability under Severe Weather Conditions

Route	Est. Trip Time	Algorithm Runtime
Route 1	8.31 min	15.71 sec
Route 2	45.35 min	32.19 sec
Route 3	14.33 min	14.57 sec
Route 4	198.61 min	57.13 sec
Route 5	181.13 min	70.01 sec

Table 6. The Performance Metrics for Dijkstra's when compared to Google Maps API

Performance Metrics		
Route	P_{rel} (%)	E_{rel} (%)
Route 1	82	1
Route 2	92	0
Route 3	83	1
Route 4	103	0
Route 5	106	0
Highest	**106**	**1**
Lowest	**82**	**0**

5.3 Bellman-Ford Results

When comparing the results obtained from various weight conditions applied to the Bellman-Ford algorithm and the

Table 7. The Adaptability Metrics for Dijkstra's when compared to Google Maps API

Adaptability Metrics			
Route	$A_{traffic}$	$A_{weather}$	$E_{roadtype}$
Route 1	M.C.	M.C.	M.C.
Route 2	M.C.	M.C.	M.C.
Route 3	M.C.	M.C.	M.C.
Route 4	M.C.	M.C.	M.C.
Route 5	M.C.	M.C.	M.C.
Average	Pass	Pass	Pass

Google Maps API, distinct differences in the outcomes become apparent.

Firstly, when considering the different traffic patterns that occur during different times of the day, it is evident that they vary significantly. Applying the rush hour weight condition to the algorithm during peak hours resulted in exploring different routes compared to Bellman-Ford with no special weights. This led to faster routes being computed due to the algorithm's exploration of routes with higher average speed limits. Table 9 indicates that prioritizing higher-speed road segments led to more efficient route selections, resulting in shorter ETAs compared to the standard Bellman-Ford algorithm.

Table 8. Base Performance of Bellman-Ford's Algorithm

Route	Est. Trip Time	Algorithm Runtime
Route 1	10.97 min	14.53 sec
Route 2	38.03 min	34.10 sec
Route 3	19.36 min	14.03 sec
Route 4	131.25 min	63.23 sec
Route 5	127.28 min	80.11 sec

Similarly, when considering different road types, particularly favoring faster categories such as roads with higher average speed limits, notable reductions in travel times were observed across most routes. This suggests that prioritizing higher-speed road segments led to more efficient route selections, resulting in shorter computed ETAs compared to the standard Bellman-Ford algorithm Table 10.

Considering severe weather factors when finding shorter routes with Bellman-Ford, the results showed an increase in all computed routes Table 11. This suggests that severe weather conditions significantly impacted the efficiency and routes computed when this condition was present.

Comparing these results to those computed on the same routes by the Google Maps API, but without any of the weight conditions, it becomes clear that Google Maps consistently outperforms the Bellman-Ford algorithm with custom edge weights in terms of efficiency. Google Maps achieves much better route execution times due to the data it is able to access. In terms of results, when Bellman-Ford was used with the rush hour conditions and the road type weight, the results showed that we found faster routes compared to Google Maps. This is surprising given the fact that the Google Maps API has access to much more real-time data and other factors that were not utilized in the computations for Bellman-Ford under these conditions.

Table 9. Bellman-Ford's Algorithm Adaptability under Rush Hour Conditions

Route	Est. Trip Time	Algorithm Runtime
Route 1	6.33 min	14.24 sec
Route 2	29.63 min	32.09 sec
Route 3	15.89 min	14.37 sec
Route 4	132.84 min	56.93 sec
Route 5	126.67 min	70.37 sec

When comparing the Bellman-Ford algorithm to Google Maps API, the performance and adaptability of Bellman-Ford are shown in Table 12. Bellman-Ford's performance metrics indicate that it provides a solid baseline for route planning. The P_{rel} values, which measure the relative performance compared to Google Maps, range from 82% to 106%, suggesting that Bellman-Ford generally finds average routes compared to Google Maps. It remains fairly close, which is acceptable for an algorithm not optimized with real-time data inputs. However, the E_{rel} lacks significantly, and Bellman-ford is magnitudes less efficient than Google Maps API algorithms.
The adaptability of Bellman-Ford, as shown in Table 13, maintains consistent Meets Conditions (M.C.) across various conditions like traffic, weather, and road type changes. The benchmarks overwhelmingly pass, showing A_{Pass} for all conditions. This indicates that while Bellman-Ford may not be as efficient when not optimized, it can handle dynamic changes in the route conditions fairly well. This adaptability is crucial for practical applications where conditions can change unexpectedly. The ability of Bellman-Ford to adapt without real-time data inputs shows its robustness and utility in scenarios where consistent data feed might not be available

5.4 A* Results

When A* is executed without a heuristic and weight, it closely mimics the behavior of Dijkstra's algorithm, generating similar route paths and total travel times. The minor change in results observed, with most times falling within a minute of each other, are expected since A* essentially reduces to Dijkstra's when no heuristic or additional weights are applied. This serves as a crucial baseline for our analysis.

Table 10. Bellman-Ford's Algorithm Adaptability under Road Type Penalty

Route	Est. Trip Time	Algorithm Runtime
Route 1	6.3 min	14.15 sec
Route 2	29.19 min	33.63 sec
Route 3	15.47 min	14.4 sec
Route 4	132.76 min	60.17 sec
Route 5	129.96 min	74.38 sec

Table 11. Bellman-Ford's Algorithm Adaptability under Severe Weather Conditions

Route	Est. Trip Time	Algorithm Runtime
Route 1	7.5 min	14.41 sec
Route 2	40 min	34.26 sec
Route 3	14.33 min	14.68 sec
Route 4	213.8 min	61.11 sec
Route 5	203.13 min	75.93 sec

Table 12. Performance Metrics for Bellman-Ford Compared to Google Maps API

Performance Metrics		
Route	P_{rel} (%)	E_{rel} (%)
Route 1	82	1
Route 2	92	0
Route 3	83	1
Route 4	103	0
Route 5	106	0
Highest	**106**	**1**
Lowest	**82**	**0**

Table 13. The Adaptability Metrics for Bellman-ford when compared to Google Maps API

Adaptability Metrics			
Route	$A_{traffic}$	$A_{weather}$	$E_{roadtype}$
Route 1	M.C.	M.C.	M.C.
Route 2	M.C.	M.C.	M.C.
Route 3	M.C.	M.C.	M.C.
Route 4	M.C.	M.C.	M.C.
Route 5	M.C.	M.C.	M.C.
Average	Pass	Pass	Pass

However, when comparing A* to the Google Maps API, it's evident that Google's service outperforms A* significantly. This

disparity in performance can be attributed to the comprehensive datasets and advanced algorithms Google leverages, which are not available in our A* implementations.
On the other hand, when A* is augmented with the Chebyshev distance heuristic, more optimal routes are produced in terms of travel time Table 14. This underscores the effectiveness of employing unique heuristics in route computation, similar to those employed by Google Maps. Surprisingly, while the Chebyshev distance heuristic improved the quality of routes, it did not lead to a notable improvement in execution time.
Overall, these findings highlight the importance of incorporating different heuristics in route planning to see different outcomes through different calculations that are unique to a heuristic. Table 15 shows the benchmark metrics on the A* algorithm. According to the relative performance and relative efficiency against the Google Maps API, the A* algorithm demonstrated superior performance. Regarding relative performance, the A* algorithm outperformed Google Maps, with the highest P_{rel} value reaching 142% compared to Google Maps, indicating it found faster routes than the control group. However, the algorithm's relevant efficiency was similarly lackluster compared to Dijkstra and Bellman-ford. One theory the research team created is that Google Maps API has optimized cashing and distributed systems, giving it an edge over unoptimized pathfinding solutions. Table 16 highlights that the adaptability metrics for the A* algorithm are not available (N/A). This data is omitted since the algorithm does not incorporate adaptability to real-time changing road conditions such as traffic, weather, and road types, a feature that is typically integral to navigation services like Google Maps. This highlights a potential limitation of the A* algorithm for real-world applications, where the dynamic edge weights and algorithm had issues during the experimental phase.

Table 14. Base Performance of A* Algorithm with Chebyshev heuristic

Route	Est. Trip Time	Algorithm Runtime
Route 1	6.33 min	13.02 sec
Route 2	29.63 min	31.43 sec
Route 3	15.89 min	15.28 sec
Route 4	132.74 min	56.07 sec
Route 5	126.67 min	72.19 sec

Table 15. Performance Metrics for A* Compared to Google Maps API

Performance Metrics		
Route	P_{rel} (%)	E_{rel} (%)
Route 1	142	1
Route 2	118	0
Route 3	100	1
Route 4	101	0
Route 5	106	0
Highest	**142**	**1**
Lowest	**100**	**0**

Table 16. The Adaptability Metrics for A* when compared to Google Maps API

Adaptability Metrics			
Route	$A_{traffic}$	$A_{weather}$	$E_{roadtype}$
Route 1	*.	*	*
Route 2	*	*	*
Route 3	*	*	*
Route 4	*	*	*
Route 5	*	*	*
Average	N/A	N/A	N/A

6 Conclusion

6.1 Effectiveness of the Proposed Algorithm

After reviewing the overall relative performance P_{rel}, relative efficiency E_{rel}, and adaptability A, the research group can conclude that the algorithms and proposed solutions partially meet the objectives. The overall relative performance P_{rel} was exceptional across the five scenarios, typically matching or exceeding Google Maps (80% to 150%). This highlights the effectiveness of the proposed algorithms when compared to the control Google Maps, often finding paths that were upwards of 25%-50% faster. That being said, Google Maps could be accounting for real-time data in their undisclosed process, but the proposed algorithms are at least comparable or better in their pathfinding capabilities. Implementing the heuristic A* algorithm and using the Chebyshev distance metric significantly enhanced the routing algorithms' performance, reaching a study-high relative performance of 142%. This method provided more accurate cost estimations from the current node to the goal, enabling the algorithms to find the most efficient paths under various conditions.
Overall, the relative efficiency of the developed algorithms E_{rel}, which measured the experimental algorithms' runtime compared to Google Maps execution speed, was extremely low (<2%). This was uniform across the results section, with A*, Dijkstra, and Bellman-ford, significantly underperforming. The underwhelming performance of the routing algorithms in the study can be attributed to several factors. Most importantly, the complexity of the algorithms might have disrupted their performance. Compared to Google Maps, which benefits from years of retooling and vast data inputs, the tested algorithms could not process and adapt to varied inputs with the same efficiency. Especially with the lack

of cache and the implementation in a high-level language. Lastly, the dynamic edge weight adjustments, which incorporated penalties and heuristics to reflect real-world conditions, did not always yield efficient route suggestions. This was particularly evident under unusual or extreme conditions, where the algorithms failed to perform as anticipated. The condition weights that were used in running with the algorithms were also able to adapt very well. The dynamic edge weights integrated into the algorithms were highly effective across multiple routing methods. In the face of changing road conditions, the algorithms found optimal routes with accurate estimated trip times with similarly exceptional adaptability A, which is discussed later in this section.
The dynamic edge weight adjustments utilized penalties and heuristics effectively to account for changing road conditions, enabling the algorithms to deliver robust and adaptable routing solutions. When challenged with severe weather, road closures, road type constrictions, and traffic, Dijkstra and Bellman-Ford were able to change edge weights dynamically and adapt to an unexpected condition in a realistic way. This led to a 100% pass rate for A, the adaptability of each algorithm. Through these adaptations, the routing algorithms demonstrated optimal performance even through constraints and challenges, marking a substantial advancement in the proposed small-scale routing technology. The penalties applied to simulate scenarios that might typically require real-time data adjustments in systems like Google Maps allowed the algorithms to adapt routes proactively rather than reactively. These weight conditions were specifically designed to represent plausible real-world events, offering a profound insight into how such factors could impact routing decisions. This approach confirmed the algorithms' capability to handle dynamic and unpredictable environments and highlighted their potential to improve route optimization significantly in real-world applications. It is worth noting, however, that the A value could not be measured and was rendered inconclusive for the A* algorithm with applied heuristics.

6.2 New Understanding of Mapping Services

Since the research group developed a practical, small-scale model, there is a new understanding of how various conditions affect routes, such as weather conditions, rush hour traffic, and using different road types. This understanding highlights how routes are updated using real-time data. Google Maps has the ability to access a very large network of real-time data, consistently flowing into its servers, allowing routes to be updated instantly. The algorithms, heuristics, and weights Google Maps uses to compute these routes are still unknown. However, the results shown here provide a good baseline for understanding the theory behind their computations. The benchmark also confirms the power of Google Maps API, which can efficiently route in milliseconds. That efficiency level can only be achieved with large, distributed systems and caching, which our small-scale model did not implement.

6.3 Future Extensions

The results show a performance gap between our algorithms, edge weight conditions, and industry-leading services, which makes it clear there is still room for improvement. Although there are best practices to follow, it's important to state that there are trade-offs when developing an algorithm with custom edge weight conditions. The experimental constraints for the algorithms defined in this report highlight the pros and cons of working with different shortest path-finding algorithms. Any future iterations of this model or experiment can implement real-time information, machine learning, caching, and algorithm computing to achieve a better-performing model.

7 Appendix

7.1 Python Libraries

NetworkX, OSMnx, Google Maps, and Plotly each offer unique tools for different aspects of data handling and visualization. NetworkX is great for working with complex networks, providing easy ways to manipulate relationships and analyze their properties [5]. OSMnx focuses on geographical and urban data, pulling street networks directly from OpenStreetMap to help with urban planning and geospatial analysis [2]. Google Maps is widely recognized for its comprehensive mapping services, offering everything from street views to real-time traffic updates, making it indispensable for applications needing navigational data [10]. Plotly excels in creating interactive and visually appealing charts, ideal for presenting data in an engaging way [6]. Together, these libraries equip users to tackle a wide range of tasks, from network theory to interactive data presentation.

7.2 Team Contributions:

Keegan's Group Contribution: Wrote everything in Experiment, Introduction, and Background. Wrote part of Benchmark script. Sources: 2, 3, 4, 7, 8, 9, 13, 14
Anthony's Group Contribution: Wrote code for the different weight conditions, algorithms and heuristics,Psudeocode, Results and Conclusion. Sources: 1, 5, 6, 10, 11, 12, 15, 16

7.3 AI Content Notice

No AI was used to source information or enhance this document.

References

[1] Samah W.G. AbuSalim, Rosziati Ibrahim, Mohd Zainuri Saringat, Sapiee Jamel, and Jahari Abdul Wahab. 2020. Comparative Analysis between Dijkstra and Bellman-Ford Algorithms in Shortest Path Optimization. *IOP Conference Series: Materials Science and Engineering* 917, 1 (sep 2020), 012077. https://doi.org/10.1088/1757-899X/917/1/012077

[2] Geoff Boeing. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65 (2017), 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[3] Siddharth Das. 2017. A fast, efficient technique for finding a path through multiple destinations. In *2017 IEEE International Conference on Electro Information Technology (EIT)*. 404–409. https://doi.org/10.1109/EIT.2017.8053395

[4] Google LLC. 2024. Google Maps Platform Documentation. https://developers.google.com/maps. Accessed: 2024-03-15.

[5] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. NetworkX. https://networkx.github.io/ Python software for complex networks.

[6] Plotly Technologies Inc. 2015. Plotly: A graphing library for making interactive, publication-quality graphs online. https://plotly.com/python/ Python library for interactive graphing.

[7] Xurui Jing and Xiaojun Yang. 2018. Application and Improvement of Heuristic Function in A* Algorithm. In *2018 37th Chinese Control Conference (CCC)*. 2191–2194. https://doi.org/10.23919/ChiCC.2018.8482630

[8] Daniel R. Lanning, Gregory K. Harrell, and Jin Wang. 2014. Dijkstra's algorithm and Google maps. In *Proceedings of the 2014 ACM Southeast Regional Conference* (Kennesaw, Georgia) *(ACM SE '14)*. Association for Computing Machinery, New York, NY, USA, Article 30, 3 pages. https://doi.org/10.1145/2638404.2638494

[9] Lingxiao Li, Muhammad Aamir Cheema, Hua Lu, Mohammed Eunus Ali, and Adel N. Toosi. 2022. Comparing Alternative Route Planning Techniques: A Comparative User Study on Melbourne, Dhaka and Copenhagen Road Networks. *IEEE Transactions on Knowledge and Data Engineering* 34, 11 (Nov 2022), 5552–5557. https://doi.org/10.1109/TKDE.2021.3063717

[10] Google LLC. 2021. Google Maps Platform. https://cloud.google.com/maps-platform/ Comprehensive mapping service including API for developers.

[11] Dennis Luxen and Christian Vetter. 2011. Real-time routing with OpenStreetMap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Chicago, Illinois) *(GIS '11)*. Association for Computing Machinery, New York, NY, USA, 513–516. https://doi.org/10.1145/2093973.2094062

[12] Heeket Mehta et al. 2019. Google Maps. *International Journal of Computer Applications* 178, 8 (May 2019). https://www.researchgate.net/profile/Pratik-Kanani/publication/333117435_Google_Maps/links/5eb6cc7da6fdcc1f1dcb10aa/Google-Maps.pdf

[13] Andreea-Cristina Rădăcină, Mihai-Bogdan Geamănu, Alexandru-Ionuţ Mustaţă, and Cătălin Negru. 2023. A Broad Analysis of Map Routing Applications. In *2023 24th International Conference on Control Systems and Computer Science (CSCS)*. 557–564. https://doi.org/10.1109/CSCS59211.2023.00094

[14] Dominik Schultes et al. 2009. Engineering Route Planning Algorithms. Karlsruhe Institute of Technology, Universität Karlsruhe (TH). https://i11www.iti.kit.edu/extra/publications/dssw-erpa-09.pdf

[15] Piyush Udhan, Akhilesh Ganeshkar, Poobigan Murugesan, Abhishek Permani, Sameep Sanjeeva, and Parth Deshpande. 2022. Vehicle Route Planning using Dynamically Weighted Dijkstra's Algorithm with Traffic Prediction. https://doi.org/10.48550/arXiv.2205.15190

[16] N Zafar and I Ul Haq. 2020. Traffic congestion prediction based on Estimated Time of Arrival. *PLoS ONE* 15, 12 (2020), e0238200. https://doi.org/10.1371/journal.pone.0238200