Analyzing the various areas within Maastricht using a Java-Based Routing Engine

Amir Mohseni

Andreas Ilia

Dimitrios Tsiplakis

Mikle Kuzin

Patrick Van Maele

Stav Cohen Shwartz

Tadiwanashe Matara

The Department of Advanced Computing Sciences

Maastricht University

Maastricht, The Netherlands

Abstract

In this report, we will explore the development of a Java-based routing program for the city of Maastricht, Netherlands. To achieve this, we used Dijkstra's and A* algorithms to compute optimal routes for walking, biking, driving, and bus travel. This approach aims to offer practical route suggestions. We conducted several experiments to test our program’s capabilities and measure how accurate and efficient our algorithms are compared to state-of-the-art routing applications. Further, we aim to assess accessibility in urban environments by evaluating socio-economic factors through an accessibility scoring system. By weighing amenities relative to postal codes, we can grade and evaluate them using a heuristic measurement. This allows us to estimate points of interest better and improve our understanding of urban accessibility. Throughout the report, we will be discussing our methodology, implementation, experimentation, and potential limitations of our approach with our routing application while addressing our research questions.

1 Introduction

Efficient transportation systems are crucial for enhancing socio-economic accessibility in urban areas. By reducing travel times, workers can reach their workplaces faster, delivery services can reach their destination quicker and first responders can attend emergencies without delay. However, the complexity of urban transport networks often makes it challenging to identify the shortest routes due to the vast number of possible ways to reach a destination. This issue can be addressed using advanced computer algorithms capable of computing optimal routes with high accuracy.

In this project, we aim to develop a Java-based routing engine designed for Maastricht's public transport system. This engine will integrate geographical data, public transport information, and socio-economic metrics to analyze and improve accessibility across different neighborhoods. By examining how transport infrastructure affects access to socioeconomic opportunities like healthcare, education, and employment, we seek to identify areas needing improvement and help make city planning fairer for everyone. The main research question of the project is: How to develop a navigation application to find the shortest route and travel time between two postal codes? with the sub-questions:

- What criteria should be used to create a comprehensive socio-economic accessibility metric to accurately score a given postal code?
- How can we integrate bus route data into our database and combine it effectively with our routing algorithm to reflect real-world public transport?

By addressing these questions, this project aims to develop an efficient, open-source routing engine that not only improves navigation within Maastricht but also provides valuable insights into the city's socio-economic landscape.

2 Methods

This section outlines our methodology. It describes the routing algorithms, transfer handling, and scoring system for neighborhoods.

2.1 Routing for Non-Public Transport

Given two postal codes alongside their longitude, latitude, and means of transportation, we needed to find a way to calculate the distance, route, and time of travel from one code to the other.

2.1.1 Straight Distance

Before implementing a complicated algorithm we created a simple function that implemented the Haversine Formula[3,4]. In simple terms, it takes two points on a sphere and, using their coordinates, calculates the distance of the straight line that connects them without penetrating the sphere. This approach produces the shortest distance possible, allowing us to test different algorithms' accuracy and validity.

$$d = 2r.sin^{-1}(\sqrt{sin^{2}(\frac{\phi_2-\phi_1}{2}) + cos(\phi_1).cos(\phi_2).sin^{2}(\frac{\psi_2-\psi_1}{2})})$$

2.1.2 Dijkstra's Algorithm

To calculate routes for non-public transport modes such as walking, biking, and driving, we utilized Dijkstra's Algorithm. This algorithm ensures the most efficient pathfinding within our dataset. We created a graph using our dataset which contains a list of nodes, each representing different locations in Maastricht with their coordinates, reachability, and accessibility information. In this graph, two nodes are connected if they are both within close proximity of each other, with a threshold set to 50 meters for our routing application, and there are no barriers between them according to the dataset.

2.2 Bus Routing

To add bus routing to our application, we utilized the GTFS dataset provided for the Netherlands, filtering the data specific to Maastricht, by looking at the entries that contain the word 'Maastricht' in the stop name and removing stops that were outside of the city. Using this data, we incorporated both direct and transfer-inclusive bus routes into our system.

2.2.1 Direct Routing

Initially, we implemented direct bus routes without transfers using SQL queries to retrieve the necessary information from the dataset. Our approach involved identifying bus stops near the origin and destination points. We then used SQL queries to find the fastest route from one of the origin's nearby bus stops to one of the destination's bus stops. This method ensured efficient direct routing between two points without the need for transfers.

2.2.2 A* Algorithm

A* is an informed search algorithm or a best-first search. This means that given a starting node and a goal node, it will try to find the shortest path to that node using a cost function. It creates a tree of paths originating at the start node and extends those paths one edge at a time until the goal node is reached or all paths are explored. By using a heuristic function that contains a lower-bound estimate of the cost to reach the goal from any node, A* can effectively prioritize paths that appear to lead more directly toward the goal. The cost function in A* is represented as $f(n) = g(n) + h(n)$, where:

- $g(n)$: The cost from the start node to the current node n.
- $h(n)$: The heuristic estimate of the cost from n to the goal.

Compared to Dijkstra's algorithm, which finds the shortest path tree from a specified source to all possible nodes, the A* algorithm finds only the shortest path from a starting node to a specific goal. This is a necessary trade-off for using a specific goal-directed heuristic[3].

For our use case, we created a graph where the vertices represented bus stops in the city, and the edges represented trips between these stops. Each edge contained information such as arrival time, trip ID, route ID, and weight, indicating the travel time between stops. Additionally, we included edges between stops within the same parent station with a penalty of 1 minute. This modification enabled our routing algorithm to facilitate transfers within a station with minimal penalty, thereby improving route finding by allowing transfers both within the same stop and between nearby stops.

First, we identified the starting nodes in our algorithm. We drew a circle with a radius of 350 meters from the starting and ending postal codes and selected all bus stops within this proximity. These stops, along with the starting time, were used as the starting nodes. We then iterated over the possible end stops to find the shortest path from the starting nodes to each end stop. In our algorithm, when reaching a node, we traversed to all its neighboring nodes. If the current time for this node plus the traversal time (the *g* function) was less than the fastest time found for the neighboring node, we updated that node's time.

We employed a combination of GTFS data and straight-line distance for our heuristic function. If a stop *n* had a bus trip to the goal stop in the GTFS database, we used that trip time as our heuristic. If there were multiple trips, we selected the one with the shortest time as our lower bound. If no direct trip existed, we used the straight-line distance, converting it into a lower bound for the time taken using an average speed of 20 km/h.

Once our algorithm reached the end node, it halted, and we recreated the path taken. We then compared all candidate end nodes based on the time taken to reach them and selected the shortest one as the best route.

This approach allows multiple transfers in our journey and is much more efficient than trying to join tables and cross-referencing possible routes.

2.3 Socio-Economic Analysis

To assess the socio-economic status of postal codes based on their proximity to various amenities, we developed a heuristic-based algorithm. It scores amenities based on the weight of their distance from a given address, using Weight decay to prioritize closer amenities (see Appendix B).

The algorithm considers all amenities and based on their distance from a given postal code, and the weighting system, each postal code gets a preliminary score. Afterwards, the scores will be passed through a normalization layer through this equation :

$$100 \cdot \frac{(S - S_{min})}{(S_{max} - S_{min})}$$

This process happens individually for amenities, shops, tourist attractions, and bus stops. When the individual sums have been calculated, they are multiplied by the specified weight sets, to signify importance, and then summed together to calculate the finalized score of that particular postal code.

2.3.1 Weights for Socio-Economic Analysis

. Category Weights:

- Shop Category Weight: $W_{shop} = 0.3$
- Amenity Category Weight: $W_{amenity} = 0.4$
- Tourism Category Weight: $W_{tourism} = 0.1$
- Accessibility Category Weight: $W_{accessibility} = 0.2$

Weight values and decay constants (see Appendix A).

2.3.2 Formulas for Socio-Economic Analysis

Exponential Decay Function:

$$e^{-\lambda \cdot d_{postAddress,point}}$$

General Point of Interest Formula:

POI = [shop, amenity, tourism]

$$S_{POI}(postAddress) = \sum_{POI} \left(\frac{\sum_{poi \in POIs} (e^{-\lambda_{poi} \cdot d_{postAddress,poi}} \cdot W_{POIType})}{\sum_{poi \in POIs} e^{-\lambda_{poi} \cdot d_{postAddress,poi}}} \right)$$

Total Score:

$$S_{total}(postAddress) = \sum_{POI} \left(\frac{\sum_{poi \in POIs} (e^{-\lambda_{poi} \cdot d_{postAddress,poi}} \cdot W_{POIType})}{\sum_{poi \in POIs} e^{-\lambda_{poi} \cdot d_{postAddress,poi}}} \cdot W_{POIType} \right)$$

Individual POI equations and Exponential Normalization equation (see Appendix B)

3 Implementation

To complete some computations we utilized a database that contained all relevant content, such as bus routes, bus stops, and amenity locations. This enabled us to create algorithms using operations on tables that resulted in faster and more precise results. The routing part of our project mainly benefited from this type of algorithm.

3.1 Routing with Transfers

When looking for a route with transfers oftentimes the user will have to move from the bus stop they step off from to another close-by one, or have to transfer multiple times. This, since it is not an immediate process, should be taken into consideration by the algorithm. To control for those two scenarios, multiple offsets were put in place to discourage more complex routes.

Since the algorithm is able to account for multiple transfers a penalty has been added to each transfer that happens, of five minutes. This means that for each transfer that happens another five minutes will be added to the end time, making that option less likely to be chosen, if the count of transfers is higher than other options and the base duration the same as them.

Walking from one bus stop to another at a transfer point is also unwanted behavior and thus penalized by the algorithm due to the increase in complexity. The amount added is one minute and as the previously mentioned penalty, it gets added for each violation that happens.

A halting condition has been put in place to not consider certain routes depending on their duration. Since traveling the main diagonal of the city lasts ninety minutes by bus the maximum time that gets considered has been set to double that at one hundred and eighty (180) minutes.

3.2 Graphical User Interface (GUI)

To allow the users to interact with our algorithms and view the results in an easy-to-interpret format a Graphical User Interface (GUI) was created. Our implementation of the GUI uses the Java Swing library due to its ease of use and simple style.

On execution of the program, a map of the city of Maastricht appears on the screen with input fields for the Postal Codes that the user leaves from and arrives at. They have the option to choose the medium of travel (i.e. Foot, Bicycle, and Car) and the type of travel they want to commit. The user can travel in a straight line or with respect to the city's road system. Any combination of choices returns both the distance and the time taken to cover that distance considering the speed of the medium.

Another choice of travel is bus traversal, where the user can specify when they want to leave and whether bus transfers are allowed. If a route is found a new window will open with the information of the route (i.e. duration, bus stops visited, line number) in text form.

Finally, the user has the ability to input a postal code and have the accessibility score shown to them in the form of a detailed report, containing the shop, amenity, tourist, and total scores.

All of the different types of algorithms also have a visual representation with the use of points and lines projected on top of the image of the map. In the case of the Bus Route with transfers, the line's color is adjusted at the point where the transfer happens.

3.3 Socio-Economics Calculation

For socio-economic analysis, we use a heuristic measurement to score amenities based on their distance to a given postal address. This distance score is determined using a Weight decay technique, using a formula of $e^{-\lambda.r}$, where r is the distance between the postal code and the selected point, and λ is a fixed amount. This method prioritizes closer amenities while still considering those further away[8].

Our weights can be related to the framework discussed in the Statistics Canada article, where a similar approach using proximity measures and gravity models is used to assess the accessibility and influence of various services and amenities on local socio-economic conditions[7]. The Weight decay method is common in spatial analysis for modeling the decay of influence with distance, ensuring that closer amenities have a higher impact on the score, a technique also seen in geographically weighted regression models used in various socio-economic and health studies[5][6].

- Essential Shops: These include establishments crucial for daily sustenance and medical needs. Under this category, supermarkets and malls provide general provisions, while pharmacies, medical supply stores, convenience stores, bakeries, butchers, and greengrocers cater to more specific daily needs.

- Specialty Shops: These are categorized as non-essential and include shops that cater to specialized interests or luxury items. The types of specialty shops include clothing, shoes, books, electronics, sports equipment, bicycles, toys, music stores, art galleries, furniture stores, and jewelry shops.

- Essential Amenities: These are facilities that provide critical services and include hospitals, clinics, fire stations, police stations, schools, universities, childcare facilities, nursing homes, and doctor's offices.

- Community Amenities: These represent important social and cultural facilities but are not classified as essential for basic needs. This category includes places of worship, community centers, libraries, arts centers, theaters, social facilities, and town halls.

- Tourism-related Locations: These are classified based on the level of tourist attraction. High tourism locations include museums, galleries, major attractions, artworks, and zoos. Moderate tourism locations encompass hotels, hostels, guest houses, caravan sites, viewpoints, and apartments.

This classification helps in prioritizing services and infrastructure development based on the essential nature and the expected footfall of each category. The calculated distance score is then multiplied by a weight assigned to each amenity category. The final weights for different amenities are as follows:

- **Essential shops**: 0.3
- **Essential amenities**: 0.4
- **Non-essential shops**: 0.1
- **Non-essential amenities**: 0.1
- **Tourist attractions**: 0.1

4 Experiments

4.1 Travel Time Analysis

For our experimentation, we chose to compare our routing application time with Google Maps, a known application that also evaluates walking, cycling, driving, and transport. We regarded our values as the observed values and the Google Maps values as the predicted values since it is anticipated that Google Maps will perform noticeably better than our algorithm.

We will be testing each mode of travel independently to see how our results compare to those of Google Maps. Since time (duration) is more important to users than distance when evaluating the strength of our program, it will be used as the unit of measurement instead of distance.

To assess walking, cycling, and driving, we will use a paired T-test. The reason for this is that our algorithm generates times based on the same routes as the corresponding travel times from Google Maps, which signifies that our data is paired. Since we are trying to evaluate the differences between the applications, we will need to calculate the differences between each pair of times (algorithm time minus real-time), and because our sample sizes are 30 each, being relatively small, is sufficient to use a t-test. In the scenario we fail the t-test, we will use the Pearson correlation coefficient test to assess the data.

For the bus routing, we tested 150 pairs of postal codes against Google Maps results. These tests were conducted with specific filters: routes were planned for a Monday at 17:00, and any routes requiring more than 400 meters of walking to the bus stop were disregarded. The total travel time considered was from 17:00 to the arrival at the final bus stop.

4.1.1 Paired T-test:

Null Hypothesis (H_0): There is no significant difference in the values our algorithm produces in comparison to the values from Google Maps.

Alternative Hypothesis (H_1): There is a significant difference in the values our algorithm produces in comparison to the values from Google Maps.
Significance Level (α): 0.05

1. Calculate the Mean of Paired Differences ($\overline{D}$):

$$\overline{D} = \frac{\sum_{i=1}^{n} (X_i - Y_i)}{n}$$

Where X_i and Y_i are the paired observations (our algorithm time and Google Maps time) and n is the sample size.

2. Calculate the Standard Deviation of Paired Differences (s_D):

$$s_D = \sqrt{\frac{\sum_{i=1}^{n} (X_i - Y_i - \overline{D})^2}{n-1}}$$

3. Calculate the t-statistic using the formula for the paired t-test:

$$t = \frac{\overline{D}}{\frac{s_D}{\sqrt{n}}}$$

Where -

$\overline{D}$ is the Mean of Paired Differences.

s_D is the Standard Deviation of Paired Differences.

n is the number of pairs (sample size).

4.1.2 Pearson Correlation Test:

Null Hypothesis (H_0): There is no significant correlation in the values our algorithm produces in comparison to the values from Google Maps.

Alternative Hypothesis (H_1): There is a significant correlation in the values our algorithm produces in comparison to the values from Google Maps.

Using the formula for the Pearson Correlation Coefficient:

$$r = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \overline{X})^2 \sum_{i=1}^{n} (Y_i - \overline{Y})^2}}$$

If |r| is close to 1, it indicates a strong correlation.

If |r| is close to 0, it indicates a weak or no correlation.

4.1.3 Testing - Walking

After calculation, the t-statistic is 6.74 and the degrees of freedom is 29. As such, our p-value is $4.89 \cdot 10^{-8}$, which is extremely small; therefore, we reject the null hypothesis that there is no significant difference between our values and the Google Maps values. From a graphical representation of our data, we can see that our algorithm tends to produce results only a few minutes away at a calculated error percentage of 11.34% (see Figure 1). However, we can calculate the Pearson correlation coefficient to determine if there is still a strong correlation between the groups.

After calculation, we get r = 0.988768 ≃ 0.99, which indicates a very strong correlation, and therefore we accept the alternative hypothesis, so there is a significant correlation in the values our algorithm produces in comparison to the values from Google Maps.

4.1.4 Testing - Cycling

After calculation, the t-statistic is 1.47 and the degrees of freedom is 29. As such, our p-value is 0.1531438 ≃ 0.15 which is > 0.05 therefore we don't reject the null hypothesis and there is no significant difference between our algorithm's values and the Google Map's values. We can see this trend graphically in the results section (see Figure 2).

4.1.5 Testing - Driving

After calculation, the t-statistic is 13.19, and the degree of freedom is 29. After running it through an application, our p-value is $2.06 \cdot 10^{-13}$, which means we reject the null hypothesis and there is a significant difference. As such, we can perform a Pearson correlation coefficient test to see if there is at least a high correlation between the values. After calculation, we get r = 0.890128 = 0.89, which indicates a high (significant) correlation between our values and the values extracted from Google Maps (see Figure 3).

4.1.6 Testing - Bus

For the buses, after calculation, we got a p-value of $1.05 \cdot 10^{-8}$, which means there is a significant difference between our algorithm's results and the ones from Google Maps. Therefore, we chose to evaluate the buses through the Mean Absolute Percentage Error (MAPE). The result was 29.44% which indicates that our application is 70% as accurate as Google Maps. Additionally, we saw that our program tends to provide shorter times almost 70% of the time in comparison to Google Maps. This leads us to create the following hypothesis:

- **Null Hypothesis (H_0)**: The mean difference (μ) in routing times is greater than or equal to zero ($\mu \geq 0$).
- **Alternative Hypothesis (H_1)**: The mean difference in routing times is less than zero ($\mu \leq 0$).
- **Significance Level (α)**: 0.05

The value of μ we got is −7.92 minutes, and since we know to reject the null due to a significant p-value, this aligns with the interpretation that our algorithm tends to provide shorter travel times. Our algorithm returns values approximately 8 minutes shorter than those from Google Maps. We estimated that 69.54% or 70% of our samples show this trend of shorter measurements. A scatter plot is constructed for additional visual comprehension (see Figure 4)

The way we extracted values from Google Maps was through their bus option at a specified time, prioritizing travel options without much walking required. On the other hand, our algorithm has a radius that limits significant amounts of walking to a maximum of 4.17 minutes. Many of the options selected from Google Maps have 10+ minutes of walking, so we attributed some of the outliers and errors to this fact. Therefore, we chose to eliminate any error beyond 50% and filter our dataset slightly, removing a total of 35 out of 150 of our original samples. This led to a reduced MAPE amounting to 18.72%, which is around 10% less than the unfiltered version, and the estimate for shorter cases percentage only dropped by 1.5% to 68.10%.

4.2 Performance Testing

For the performance testing, we implemented a timer to record the runtime of 150 random pairs of postal codes and calculated the average. The code is provided in the appendix (see Appendix F). We set the time to 8:00 AM as the standard time for all trips in the experiment since 8:00 AM is at the times when bus traffic is heaviest in Maastricht.

5 Results

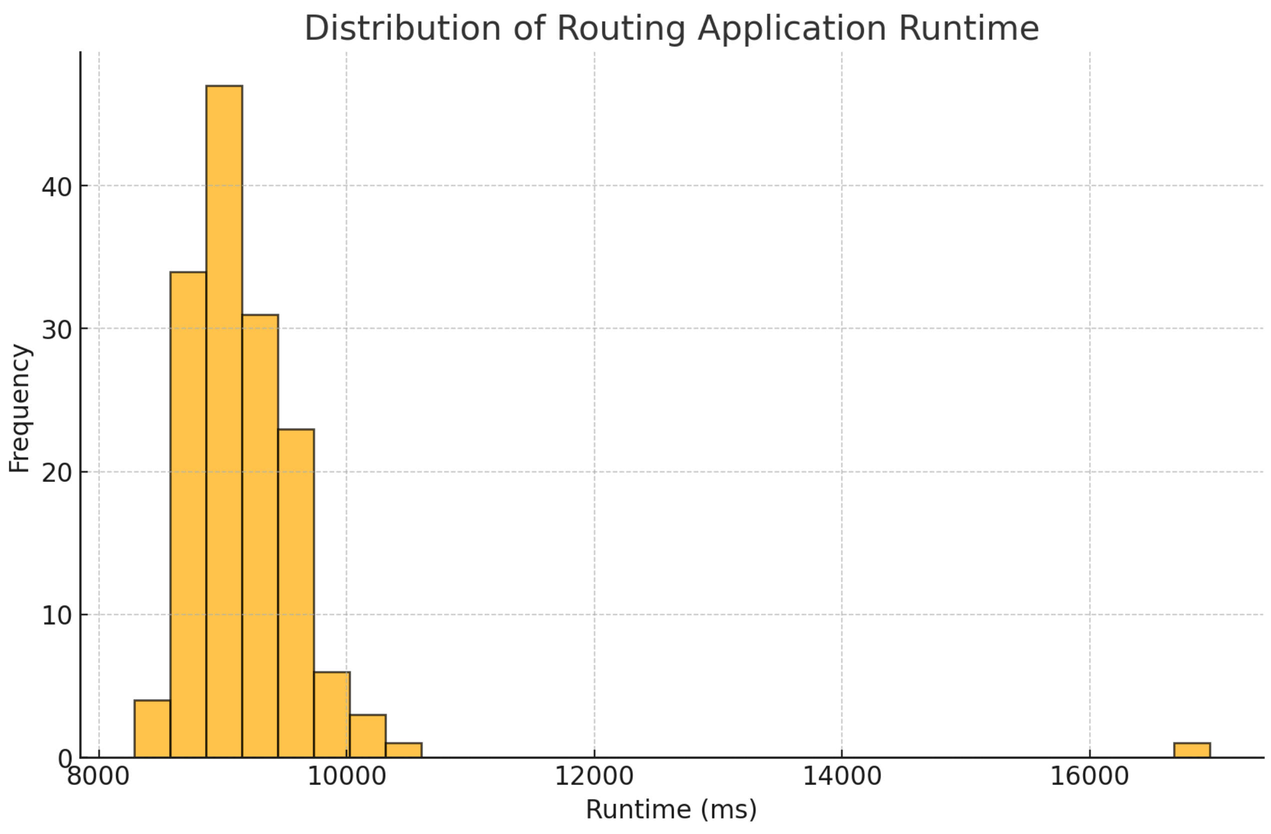
5.1 Travel Time Analysis

Figure 1-4 (see Appendix C).

5.2 Performance Metrics

- **Count**: 150
- **Mean Runtime**: 9205.48 ms
- **Standard Deviation**: 738.89 ms

- **Minimum Runtime**: 8285 ms
- **Maximum Runtime**: 16970 ms



5.3 Socioeconomic Scores

Based on the heuristic-based algorithm that we used for socioeconomic measures, we got the following results (see Appendix D)

Average score by neighborhood bar charts (see Appendix E)

6 Discussion

The results and their visualizations now constitute a good benchmark for future improvements and help us hone in on specific improvement points.

The rejection of the null hypothesis points to our program being considerably faster at finding routes within the 350m walking distance threshold. The walking distance threshold is considerably shorter than Google Maps, as we wanted the travel time to be mainly focused on the bus routes. It is also important to point out that the higher walking distances provided by Google Maps can lead to an inflated travel time in comparison to our algorithm.

Figure 1 illustrates a scatter plot comparing the walking times predicted by our routing algorithm to those provided by Google Maps (see Figure 1). Each point on the plot represents a pair of corresponding times for the same two postal codes. The plot shows that the time difference between the two groups typically differs by only a few minutes on average. This suggests that our algorithm may perform comparably to Google Maps in estimating walking times.

Figure 2 illustrates a smooth line chart comparing cycling times between our algorithm and Google Maps (see Figure 2). The chart showcases the strong correlation between the two sets of times. The chart helps to visualize trends and patterns in the data, which allows us to see that our algorithm closely follows the cycling time estimates of Google Maps, suggesting that our routing algorithm effectively calculates cycling routes and provides reliable estimates.

Figure 3 illustrates a smooth line chart comparing driving times between our algorithm and Google Maps (see Figure 3). This chart represents the differences in values between the two data sets. Using the chart, we can clearly compare trends in our data and highlight systematic deviations and the tendency to provide shorter times. Through analysis, we can identify that our algorithm may need more work so that it could better match the driving time estimates of Google Maps.

Figure 4 illustrates a scatter plot comparing the bus travel times produced by our routing algorithm to those provided by Google Maps (see Figure 4). This plot is particularly useful for visualizing the behavior of the data in relation to y = x. Points that fall directly on this line indicate perfect agreement between the two sets of times, while points above or below the line indicate overestimation or underestimation, respectively. The scatter plot allows us to identify patterns and potential discrepancies in bus travel time estimates, offering insights into how our algorithm can be improved.

As for the runtime, we can confidently assert that our program finds a trip on average in 9.31 seconds, with times ranging from around 8.2 seconds (8, 200 ms) to 16.97 seconds (16, 970 ms). We found no significant direct correlation between the postal codes and the runtime. Graph 4 depicts the 16.97 seconds edge case, which is to be expected as this correlates to the first run of our program when the graph is first generated, which we found took around 6 seconds. After the graph is generated, all other trips are run with the same graph, so the runtimes significantly drop.

7 Limitations

Our research faced several limitations that impacted the overall results. The computational environment that we used, influenced the performance of our routing algorithms, and the limited data for calculating the socio-economic score affected the precision of our results. Additionally, the fact that the database that we use is online disallows the user from accessing the application when no internet access is present

7.1 Routing

The performance and accuracy of the algorithms depended on the quality of the OpenStreetMap data, which may not be as reliable as commercial mapping services. Additionally, the computational environment and hardware used could affect the results, indicating a need for further testing under varied conditions to validate our routing program's effectiveness.

7.1.1 Finding Bus Stops Nearby

Given a postal code, we need to find nearby bus stops to use as candidate stops for our routing. A radius of 350 meters has been established to limit the number of bus stops considered, thereby reducing the number of routes that pass through these stops. This optimization enhances the speed of execution and reduces the complexity of our program. However, if there are no bus stops within the radius, we will select the closest available stop. While this approach improves efficiency, it may not always yield the optimal outcome.

7.1.2 Bus Transfers

Due to how the transfers are identified, it is inefficient to consider bus stops to transfer to that are not the same bus stop or not one in the same parent station (approximately a one-minute-long walk away), due to our limited computational resources.

7.2 Socio-Economic

Even though our current approach in calculating the socio-economic score of postal codes in the city of Maastricht uses all the data that has been given, which is the types and positions of different amenities, the inclusion of new data, such as the average household salary, job opportunities, and cleanliness would grant us a more precise measure of that value[4].

8 Conclusion

In our study, we created and tested an efficient routing program to determine the shortest paths between postal codes in Maastricht, utilizing open-source data from OpenStreetMaps. Initially, we used the Haversine Formula for baseline distance calculations, then implemented Dijkstra's Algorithm and the A* Algorithm for more precise pathfinding.

Additionally, our accuracy assessments, which compared routes generated by our algorithms with those from commercial mapping services like Google Maps, demonstrated that our open-source-based solutions are competitive in terms of precision. This serves as an example of the potential of heuristic-based algorithms and open-source tools as cost-effective and efficient alternatives for route planning across various sectors.

The conclusion of our research highlights the significant socio-economic benefits of implementing efficient routing algorithms. By optimizing transportation networks and reducing travel times, our program can enhance access to essential services, improve the efficiency of logistics operations, and contribute to overall urban development. These advancements support better urban planning and policy-making, ultimately fostering socio-economic growth and improving the quality of life in urban areas.

References

[1]Kettle, S. (2017, May 10). Distance on a sphere: The Haversine Formula. Esri. https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128

[2]Chopde, N. R., & Nichat, M. K. (2013). Landmark based shortest path detection by using A* and Haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering, 1*(2), 298.

[3] Russell, S. J., & Norvig, P. (2018). *Artificial intelligence: A modern approach* (4th ed.). Boston, MA: Pearson. ISBN 978-0134610993. OCLC 1021874142.

[4] Kaufman, Sarah, Mitchell L Moss, Justin Tyndall, and Jorge Hernandez. 2014. “Mobility, Economic Opportunity and New York City Neighborhoods.” NYU Wagner Research Paper, no. 2598566.

[5]Gao, J., & Li, S. (2011). Detecting spatially non-stationary and scale-dependent relationships between urban landscape fragmentation and related factors using geographically weighted regression. *Applied Geography, 31*(1), 292-302. https://doi.org/10.1016/j.apgeog.2010.06.003

[6]Jaimes, N. B. P., Sendra, J. B., Delgado, M. J., & Plata, R. F. (2010). Exploring the driving forces behind deforestation in the state of Mexico (Mexico) using geographically weighted regression. *Applied Geography, 30*(4), 576-591. https://doi.org/10.1016/j.apgeog.2010.05.001

[7]Statistics Canada. (2020). Measuring proximity to services and amenities: An experimental set of indicators for neighbourhoods and localities. Retrieved from https://www150.statcan.gc.ca/n1/pub/18-001-x/18-001-x2020001-eng.htm

[8]Elucidating the spatially varying relation between cervical cancer and socio-economic conditions in England. (2021). *International Journal of Health Geographics*. https://ij-healthgeographics.biomedcentral.com/articles/10.1186/s12942-021-00259-4

Appendix

Appendix A.

1. Shop Weights

- Essential Shop Weight: $W_{essentialShop} = 0.5$
- Specialty Shop Weight: $W_{specialtyShop} = 0.3$
- Miscellaneous Shop Weight: $W_{miscShop} = 0.2$

2. Amenity Weights

- Essential Amenity Weight: $W_{essentialAmenity} = 0.5$
- Community Amenity $W_{communityAmenity} = 0.3$
- Recreational Amenity $W_{recAmenity} = 0.2$

3. Tourism Weights

- High Tourism Weight: $W_{highTourism} = 0.5$
- Moderate Tourism Weight: $W_{moderateTourism} = 0.3$
- Low Tourism Weight: $W_{lowTourism} = 0.2$

4. Decay Constants

- Essential Shop Decay: $\lambda_{essentialShop} = 0.3$
- Specialty Shop Decay: $\lambda_{specialtyShop} = 0.5$
- Miscellaneous Shop Decay: $\lambda_{miscShop} = 0.7$
- Essential Amenity Decay: $\lambda_{essentialAmenity} = 0.3$
- Community Amenity Decay: $\lambda_{communityAmenity} = 0.5$
- Recreational Amenity Decay: $\lambda_{recAmenity} = 0.7$
- High Tourism Decay: $\lambda_{highTourism} = 0.3$
- Moderate Tourism Decay: $\lambda_{moderateTourism} = 0.5$
- Low Tourism Decay: $\lambda_{lowTourism} = 0.7$

Appendix B.

Normalization Equation

$$100 \cdot \frac{(S - S_{min})}{(S_{max} - S_{min})}$$

Where

S = Score of interest

S_{min} = minimum score

S_{max} = maximum score

Shop Scores:

$$S_{shop}(postAddress) = \frac{\sum_{shop \in shops} (e^{-\lambda_{shop} \cdot d_{postAddress,shop}} \cdot W_{shopType})}{\sum_{shop \in shops} e^{-\lambda_{shop} \cdot d_{postAddress,shop}}}$$

Amenity Scores:

$$S_{amenity}(postAddress) = \frac{\sum_{amenity \in amenities} (e^{-\lambda_{amenity} \cdot d_{postAddress,amenity}} \cdot W_{amenityType})}{\sum_{amenity \in amenities} e^{-\lambda_{amenity} \cdot d_{postAddress,amenity}}}$$

Tourism Scores:

$$S_{tourism}(postAddress) = \frac{\sum_{attraction \in attractions} (e^{-\lambda_{attraction} \cdot d_{postAddress,attraction}} \cdot W_{tourismType})}{\sum_{attraction \in attractions} e^{-\lambda_{attraction} \cdot d_{postAddress,attraction}}}$$

Appendix C.

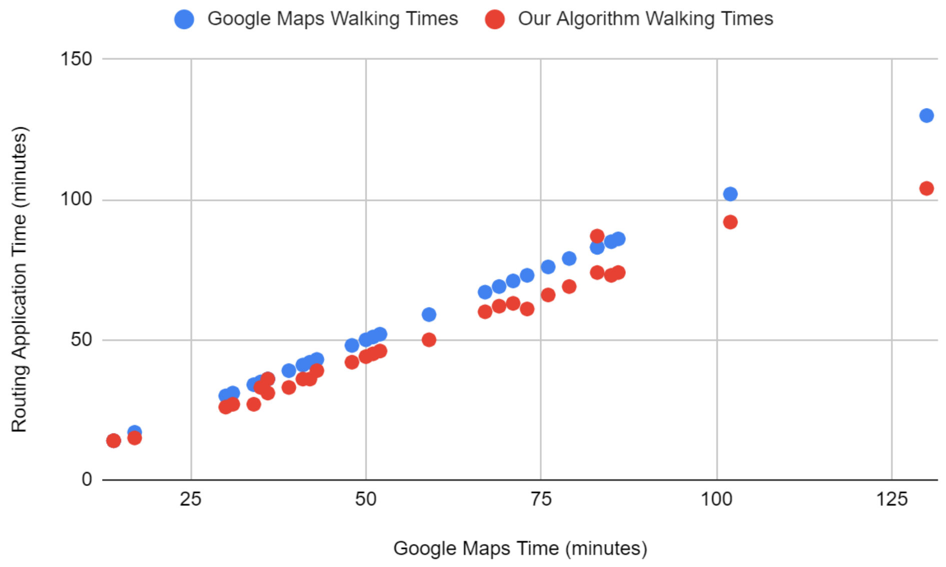


Figure 1. Google Maps Walking Times vs Our Algorithm Walking Times - Scatter Plot.

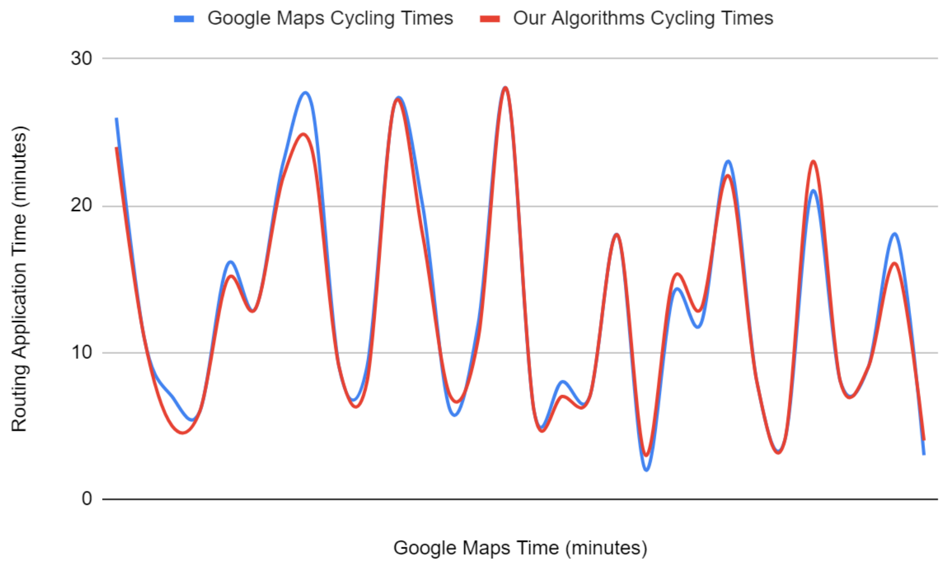


Figure 2. Google Maps Cycling Times vs Our Algorithm Cycling Times - Smooth Line Chart.

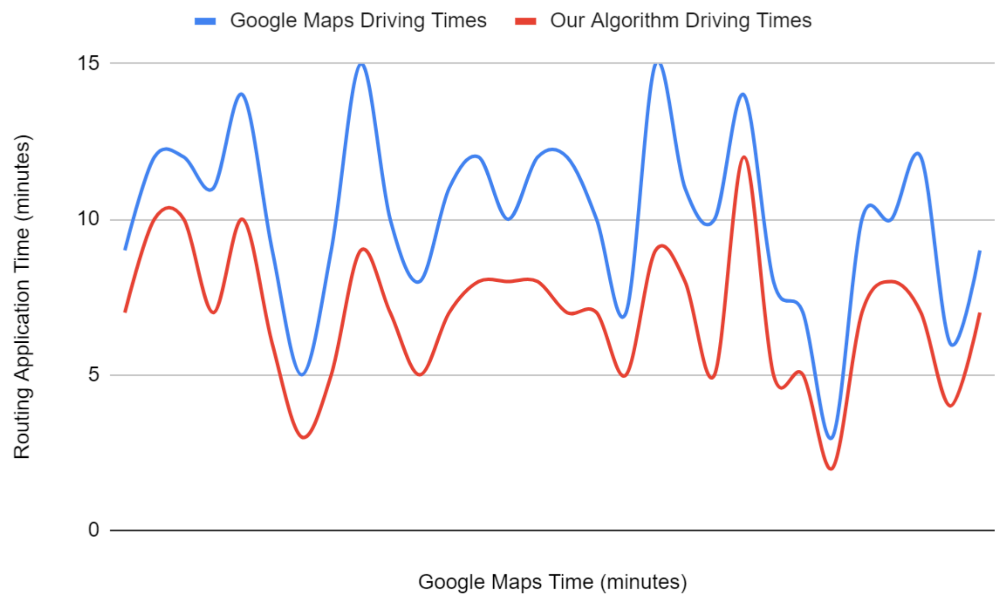


Figure 3. Google Maps Driving Times vs Our Algorithm Driving Times - Smooth Line Chart

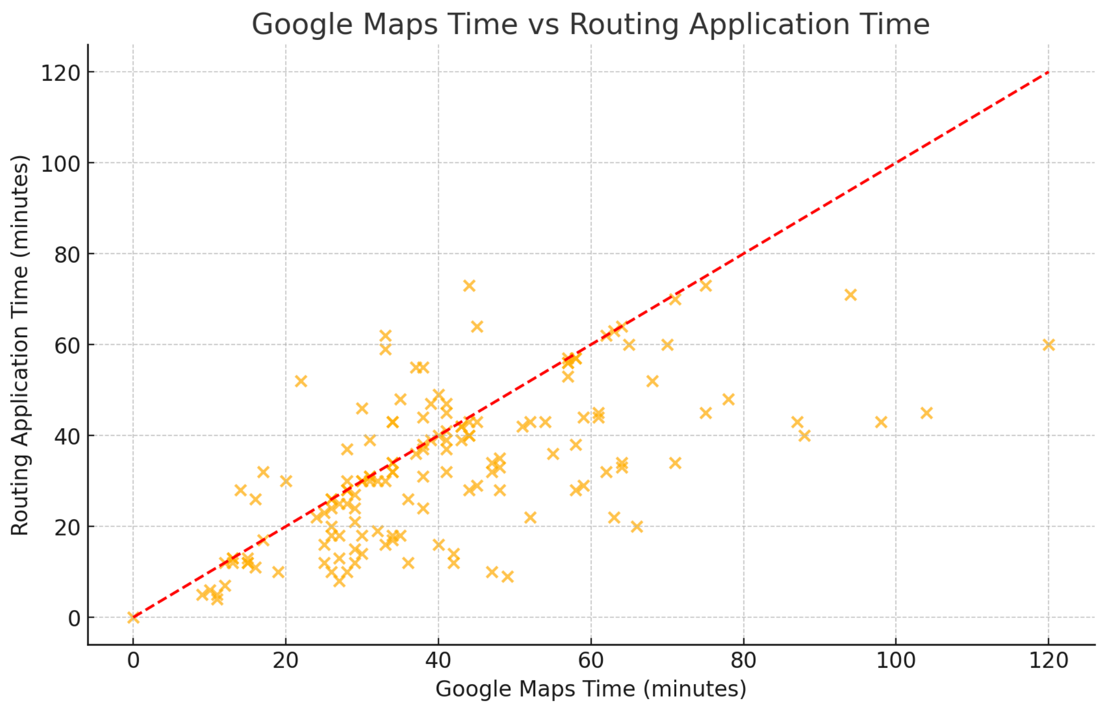


Figure 4. Google Maps Time vs Routing Application Bus Times Scatter Plot.

Appendix D.

Postal Code	Score	Amenity Score	Shop Score	Tourism Score	Accessibility Score
6211GP	100	99.92559402	99.70940304	98.74976615	94.89442578
6211EE	99.95174473	99.59515119	100	98.09160867	95.21026592
6211GR	99.93233086	99.85097474	99.57595619	98.82772777	94.87078167
6211ED	99.88661928	99.53699077	99.8781824	98.00272695	95.23223969
6211GL	99.8218947	99.83003359	99.27962215	98.67131963	94.89016278
6211EC	99.81680826	99.44800459	99.73274665	98.17515881	95.19750351

Table 1 - Highest scores recorded in our measurements

Postal Code	Score	Amenity Score	Shop Score	Tourism Score	Accessibility Score
6223HB	2.568756537	1.941952692	1.657490269	3.346211257	4.638309054
6223HD	2.351998124	1.847854151	1.484047363	3.014077382	4.182634525
6223GZ	2.297992153	1.811941121	1.445312719	2.937941336	4.084011412
6223GX	2.119917471	1.687398153	1.324619153	2.709111021	3.749425572
6223GW	1.634176618	1.309666747	1.014757953	2.127351202	2.862532298
6223HV	0	0	0	0	0

Table 2 - Lowest scores recorded in our measurements

Appendix E.

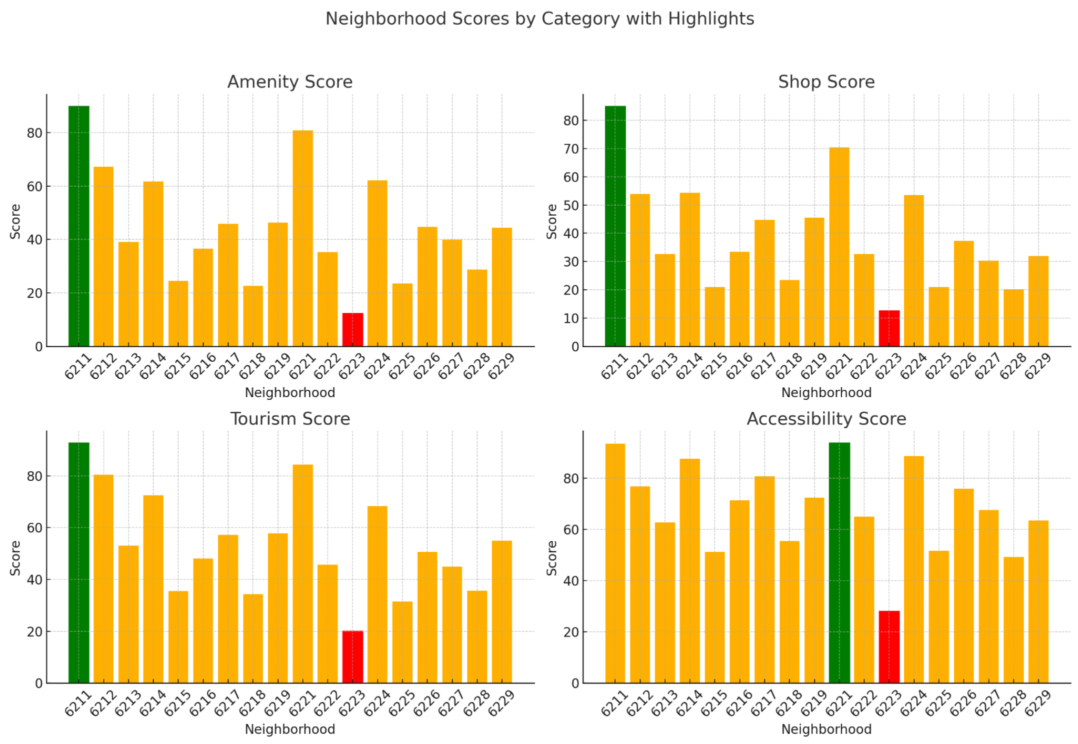


Figure 5. Neighborhood Scores by Category

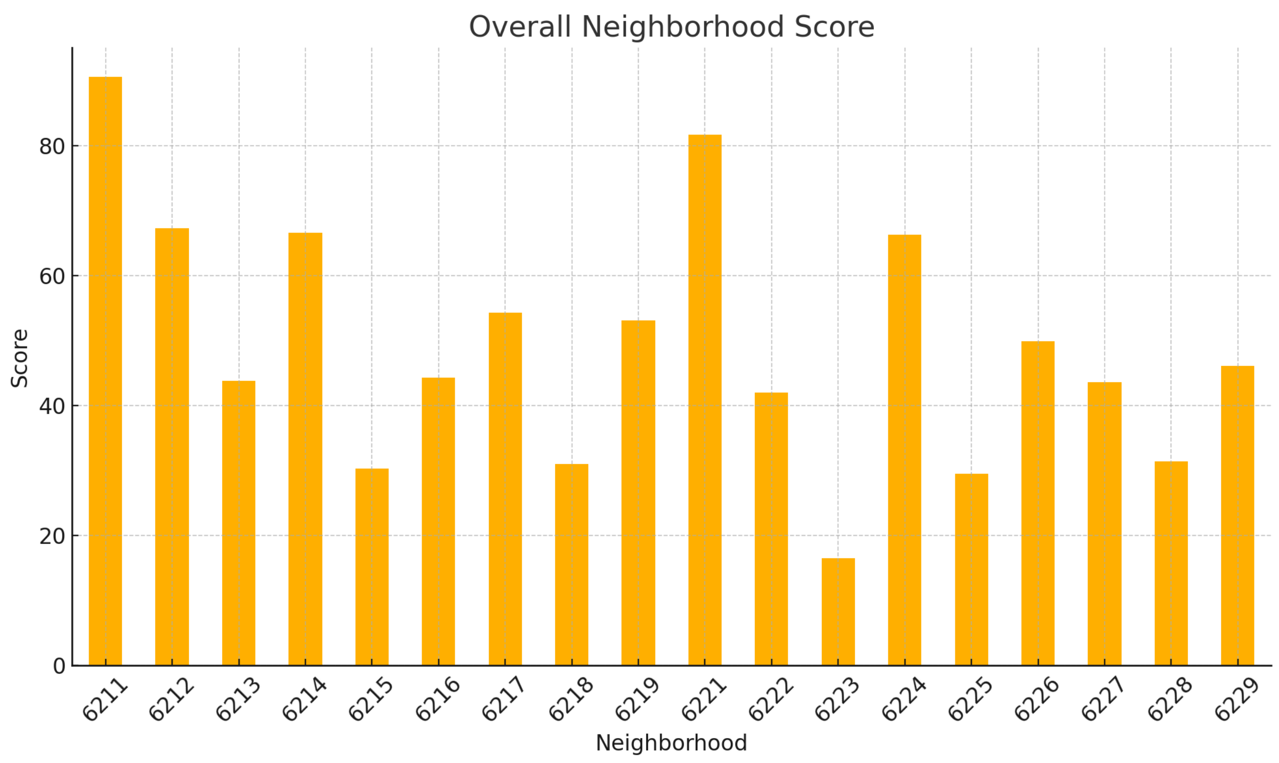


Figure 6. Overall Neighborhood Scores

Appendix F.

GitHub Repository:

- https://github.com/Amir-Mohseni/Routing-Application

Code used for generating random postal codes for the experiments:

- Testing-P1-2.ipynb

Experiment Results:

- Testing - Project 1-2

Presentation:

- 12-FinalPresentation