Project Transistor

BCS1600, Project 1-2

Group 24

Full Name	Student ID
Dario Dastan	I6339501
Marcel Pendyk	I6364063
Tim Loewen	I6343310
Calin Suconicov	I6359878
Long Luong	I6359380
Eduard Prescorniță	I6362560
João Tavares	I6366229

Examiners: Aki Härmä, Tom Bitterman

Maastricht, June 24, 2024

Abstract

This paper studies the correlation between the connectivity of a neighborhood's public transportation system with its socio-economic status in the city of Maastricht, the Netherlands. This problem was chosen because we want to investigate the problem of transport poverty in the Netherlands. We answer this question by calculating an index for each neighborhood in which the index is positively correlated to its accessibility metric. The results reveal that there is a positive relation between a neighborhood's public transportation system and its socio-economic status. These findings suggest that improving public transportation connectivity can improve socio-economic opportunities in disadvantaged neighborhoods.

1 | Introduction

When the distances are too large, individuals with limited mobility may find themselves restricted from engaging in the economic and social aspects due to decreased access to opportunities, services, and social networks [3]. This phenomenon is often referred to as "transport poverty". When transport poverty is prevalent, it implies a lack of balance of economic, social and environmental qualities [8].
In the context of Maastricht, optimizing the public transport network is essential for improving residents' quality of life [8], and ensuring that each person has equal opportunities to access public transport. Project "Transitor" aims to address these challenges by developing a Java-based routing engine to visualize public transport navigation within the city of Maastricht.
This paper provides insights into the process and methods behind the development of the public transport routing system focused on Maastricht. This paper also determines a metric that measures the accessibility metric of each postal code in Maastricht. As such, the following research question is answered: **"How does the connectivity of a neighborhood's public transportation system correlate with its socio-economic status in the city of Maastricht?"**.

2 | Methods

This section describes the data collection, the data integration, and the routing algorithm.

2.1 | Data collection

We were provided with the file `MassLatLon.xlsx`, which contains geographical information about the latitude and longitude of locations. Additionally, we were provided with an API at `http://project12.ashish.nl/get_coordinates`. This API is used if and only if a given postal code is not present in `MassLatLon.xlsx`[6]. Besides this, we were also provided with the file `Maastricht.osm.pbf`, which contains the map data of Maastricht.

Next, we were given General Transit Feed Specification (GTFS) data [6] [7], which contains information about public transportation schedules and its associated geographic information. Each file models a particular aspect of the transit system. The given GTFS data is as follows:

- agency.txt: Details about the transit agency, including name, URL, and time zone.
- stops.txt: Information about individual stops, stations, or areas where vehicles pick up or drop off passengers.
- routes.txt: Descriptions of routes in terms of long and short names, route type (bus, tram, etc.), and route ID linking to other files.
- trips.txt: Trips for each route, including trip IDs and headsigns, linked to schedules and sequences of stops.
- stop_times.txt: Times that a vehicle arrives at and departs from individual stops for each trip.
- calendar.txt or calendar_dates.txt: Service dates, showing when services are available or exceptions occur.

GeoJSON is a standard format for encoding geographic data structures, making it a useful asset for mapping and spatial analysis. We were provided the GeoJSON data of various points in Maastricht [6] [7].

- Tourism (tourism.geojson): Features 259 points of interest related to tourism like museums and historical sites.
- Shops (shop.geojson): Details about 697 shopping locations and stores, providing insights into commercial accessibility.
- Amenities (amenity.geojson): Contains 1173 features representing various public amenities and facilities.

2.2 | Data integration

The data at `MassLatLon.xlsx` was converted to the file `MassZipLatLon.csv` file. This was done to ensure that the methods in `CSVDataFetcher.java` can read the contents of the .csv file. The file `Maastricht.osm.pbf` was converted to the file `Maastricht.map`. This was done to ensure that this file can be read with the Java library MapForge.

Given the GTFS data described in the previous section, the data needs to be processed before it can be implemented. To accomodate for this, an SQLite database was created and saved under the name `database.db`. The GTFS data is processed and stored in `database.db`. This is done by calling methods described in `DB.java` and `DatabaseInit.java`. The class `DatabaseInit.java` reads the aforementioned GTFS data and populates the database `database.db`. This is done by using the `java.sql` package along with various SQL queries in the `DatabaseInit.java` class.

For each postal code in Maastricht [2], the distances to various points of interest (futher referenced as POIs) were calculated using the Haversine formula. To efficiently manage these calculations, we stored each POI under its respective category (166 categories in total) in separate K-D trees. We used the K-D trees data structures because iterating over each separate POI to calculate the aerial distance for all POIs

would be very complex and inefficient. This data structure allow us to efficiently handle multi-dimensional data, enabling quick nearest-neighbor searches [1]. This significantly reduces the computational complexity and time required to find the closest POI within each category to a given postal code, making the process much more efficient. Each tree contained the latitude and longitude of POIs within the same category. Using these trees, we performed quick nearest-neighbor searches to find the closest POI within each category to the postal code being analyzed. Once the nearest POI was identified for each category, we applied our algorithm to calculate the cost of travel time from the postal code to that POI. This process was repeated for all 166 categories, ensuring a feasible accessibility analysis for each postal code.

2.3 | Algorithms

2.3.1 | Distance calculation

Due to the geographical nature of a map, we are calculating a distance between two geographical points on the earth's surface. In order to calculate the great circle distance, the Haversine formula is used. The Haversine formula takes into account the spherical shape of earth (which calculates the aerial distance) [9]. This feature has been implemented during the first phase of the project.

2.3.2 | Pathfinding algorithms

The pathfinding algorithm can be split into two categories: non-bus traversal and bus-traversal

Non-bus traversal The built-in method `GHRequest` in GraphHopper is used to do the routing.

Bus traversal The bus traversal method uses Dijkstra's algorithm [4]. Dijkstra's algorithm is suitable for this project due to its efficiency with weighted graphs, which represent the different travel times between bus stops. Every edge is a weighted edge where the weight indicates the time duration it takes to traverse from node A to node B.

2.3.3 | Index calculation

In order to evaluate a given's postcode's accessibility index, we are creating our own algorithm that takes a postal code as input and outputs an index. The details of this algorithm are described in the next section.

3 | Implementation

This section talks about the system architecture and the different algorithms used.

3.1 | System architecture

The library GraphHopper was used as it is a Java library that provides us with a routing engine. This particular library was chosen due to its ease of integration with Java applications. On top of this, the library MapForge was used. Mapforge is a map library that provides the tools necessary to create a map application. The library is used to render the map of Maastricht and make it easier to draw on a map with coordinates. This map is used for our visualisations. Additionally, JavaFX and Swing were used for the UI components. JavaFX was used due to its compatibility with the visual layout tool SceneBuilder, which was used to create the UI components.

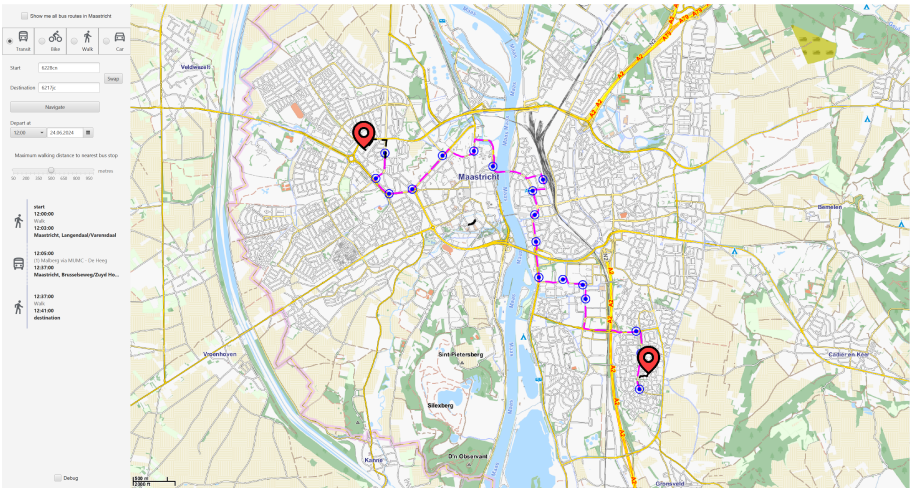

Figure 3.1: Our map application

3.2 | Algorithms

3.2.1 | Haversine Distance Calculation

The implementation of the distance calculation was based on the Haversine formula to account for the spherical shape of the Earth. The distance calculation function takes two geographical points as input and returns the distance in meters. External dependencies only included Java's `java.lang.Math` for rounding and radians conversions.

Pseudo Code

```
function distanceTo(point1, point2):
lat1 = rad(point1.lat)
lon1 = rad(point1.lon)
lat2 = rad(point2.lat)
lon2 = rad(point2.lon)

dlat = lat2 - lat1
dlon = lon2 - lon1

a = (sin(dlat / 2))^2 + cos(lat1) * cos(lat2) * (sin(dlon / 2))^2
c = 2 * atan2(sqrt(a), sqrt(1 - a))
r = 6371 // radius of earth in km
distance = r * c * 1000  // convert to meters

return round(distance, 2)
```

3.2.2 | Pathfinding Algorithm

The pathfinding algorithm can be split into two categories: non-bus traversal and bus-traversal

Non-bus traversal The pathfinding functionality was implemented using the GraphHopper library, which simplifies the integration of Dijkstra's algorithm and provides additional support for handling public transportation data. GraphHopper's configuration allowed us to import local maps and set up the graph. GraphHopper uses the file at `Maastricht.osm.pdf`. The built-in method `GHRequest` in GraphHopper is used to do the routing in our method `pathFind`. `pathFind` takes in two arguments, which are the start and end points. The method creates a routing request between the start and end using the imported map data. The best route is retrieved and displayed on the map, and the distance is returned.

Bus traversal This pathfinding algorithm uses Dijkstra's algorithm. To achieve this, the following is done:

1. Represent each bus stop as a node. Additionally, represent the starting and ending points as nodes.
2. For each bus stop, create edges (connections) to the subsequent stop(s) on the same bus line. Each edge between stops has a list of departure and arrival times to track when we can transition to the next node and when we will arrive. If an edge already exists from one node to another, add the information to the list in the existing edge. The edges are one-way.
3. For each bus stop, check if it's close enough to another stop to walk. If so, connect them with an edge representing walking time.
4. Use Dijkstra's algorithm to find the shortest path to the destination
5. Once the end node has been reached, backtrack to find the complete path taken.
6. Try to minimize the number of edges taken by saving the amount of edges taken so far to make the path more optimal

3.2.3 | Index calculations

The index calculation takes a postal code from Maastricht as an input and returns an index as its output. The value of the index is directly correlated to the accessibility of the postal code. The index calculation works as follows:

1. We input a postal code that is in Maastricht.
2. For each unique POI, we take the closest POI (aerial-distance) to the postal code travel time-wise.
3. The index of the POI is calculated using $\frac{\text{weight}}{\text{travelTime}}$.
   [a] Each POI has a unique weight assigned to it. The weights were chosen arbitarily for each POI. For the full list of weights, see the appendix.
   [b] The travel time by default uses the bus for large enough distances, otherwise it is calculated by using the walking path.
4. All the indexes of every POIs are added together. This final index is the accessibility index for the given postal code.
5. Return the final index.

4 | Experiments

Our approach of answering the research question **"How does the connectivity of a neighborhood's public transportation system correlate with its socio-economic status in the city of Maastricht?"** involves utilising the index calculation explained in the previous section.In particular, we applied the index calculation for every postal code in Maastricht. The result is an index for each postal code in Maastricht.

5 | Results

The heatmap below shows the three primary colors: red, yellow and green. The heatmap transitions using a gradient.

- Red: The accessibility value has a value of $v \in [0, 0.2)$
- Yellow: The accessibility value has a value of $v \in [0.2, 1)$
- Green: The accessibility value has a value of $v \in [1, \infty)$

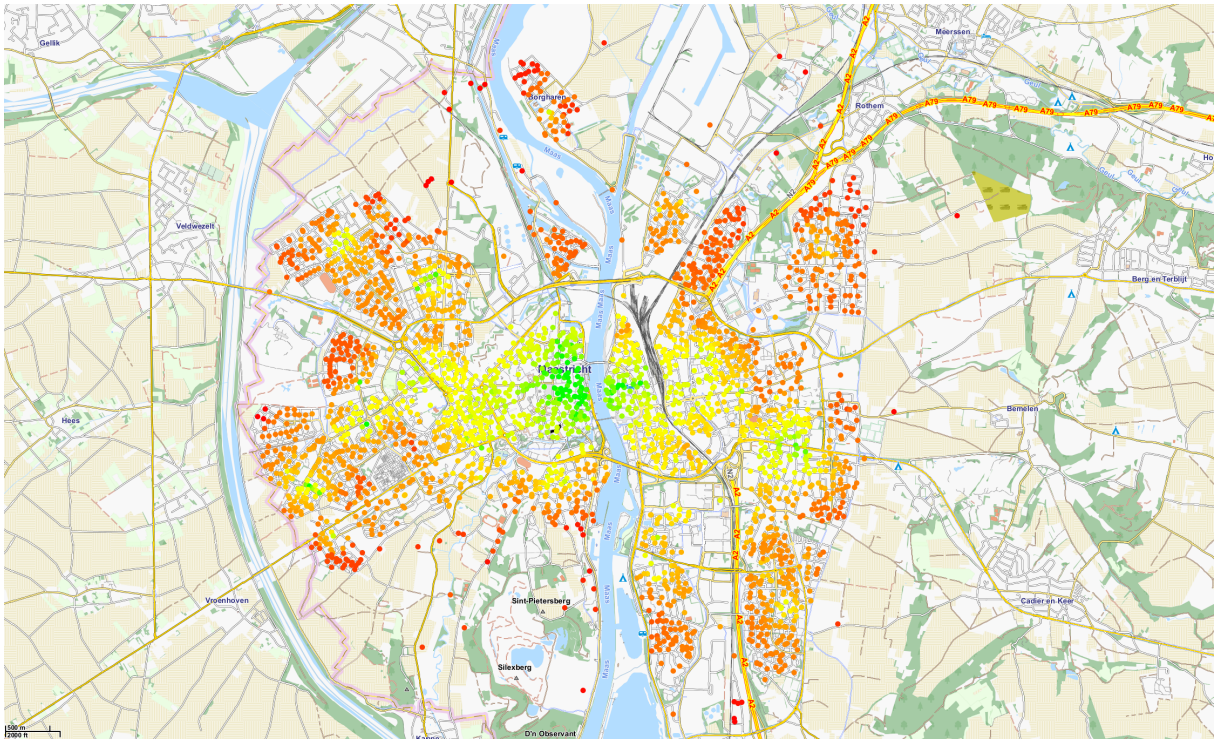

Figure 5.1: Heatmap accessibility of Maastricht at 12PM (noon)

6 | Discussion

The result in figure 5.1 shows the accessibility of various postal codes within Maastricht, with colors representing different levels of accessibility. Green dots indicate areas with high accessibility, primarily concentrated in the city center. Yellow to red dots denote progressively lower accessibility, with the outer regions experiencing the least access.

The correlation between public transport accessibility and socio-economic status is evident. Central areas, with better public transport connectivity, likely offer more socio-economic opportunities such as employment, education, and services. In contrast, areas with lower accessibility may face transport poverty, limiting residents' socio-economic prospects. This is apparent in the outer areas of Maastricht, such as in the north-east, the north, the west, and the south-east of Maastricht. This spatial distribution aligns with our research question that there is a positive correlation between public transport accessibility and its socio-economic status .

Our findings suggest that enhancing public transportation networks in less accessible areas could improve socio-economic outcomes. Our study advances the understanding of transport poverty by quantifying accessibility's impact on socio-economic status. However, our data relies on static schedules done at 12PM, which does not reflect real-time scheduling. This issue suggests the need to generate more data across different times of the day.

7 | Conclusion

By calculating an accessibility index for each neighborhood, we found a positive relationship between public transportation connectivity and socio-economic status. Our findings suggest that improving public transportation connectivity can enhance socio-economic opportunities in disadvantaged neighborhoods, which answers our research question **"How does the connectivity of a neighborhood's public transportation system correlate with its socio-economic status in the city of Maastricht?"**. This suggests that neighborhoods with better public transportation access tend to have higher socio-economic opportunities. These findings indicate the significance that public transportation plays in improving the socio-economic status in disadvantaged neighborhoods. Improved connectivity can potentially alleviate transport poverty and promote equitable urban development. Future research could focus on exploring the specific types of public transportation improvements that most effectively enhance socio-economic status. Additionally, investigating other cities could provide a better understanding of the relationship between transportation connectivity and its socio-economic status.

8 | References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[2] CBS (Central Bureau Statistics Netherlands). Tabel: Bevolking; geslacht, leeftijd en burgerlijke staat, 1 januari, 2023. Retrieved June 20, 2024.

[3] Graham Currie and Alexa Delbosc. Modelling the social and psychological impacts of transport disadvantage. *Transportation (Amst.)*, 37(6):953–966, November 2010.

[4] E W Dijkstra. A note on two problems in connexion with graphs. *Numer. Math. (Heidelb.)*, 1(1):269–271, December 1959.

[5] Malvika Dixit, Subeh Chowdhury, Oded Cats, Ties Brands, Niels van Oort, and Serge Hoogendoorn. Examining circuity of urban transit networks from an equity perspective. *J. Transp. Geogr.*, 91(102980):102980, February 2021.

[6] Department of Advanced Computing Sciences. *Project 1.2: Transitor*. Department of Advanced Computing Sciences, Maastricht, Netherlands, 2024.

[7] OpenStreetMap Contributors. Feature collections generated by overpass-ide, 2023. Retrieved June 20, 2024.

[8] Linda Steg and Robert Gifford. Sustainable transportation and quality of life. *J. Transp. Geogr.*, 13(1):59–69, March 2005.

[9] Glen Van Brummelen. *Heavenly mathematics*. Princeton University Press, Princeton, NJ, December 2012.

A | Appendix

A.1 | List of all 166 points of interests with their respective weight

RECYCLING(1.0)
POST_BOX(0.5)
LIBRARY(1.2)
POLICE(1.5)
PLACE_OF_WORSHIP(0.8)
COURTHOUSE(1.3)
FOUNTAIN(0.4)
UNIVERSITY(1.7)
RESTHOUSE(1.0)
PARKING_ENTRANCE(0.7)
PARKING(0.6)
FUEL(1.0)
BANK(1.2)
POST_OFFICE(1.0)
ATM(0.9)
INFORMATION(0.8)
SCHOOL(1.5)
PUB(0.5)
BICYCLE_PARKING(0.6)
MOPED_PARKING(0.4)
CINEMA(1.4)
RESTAURANT(1.0)
FAST_FOOD(0.7)
VENDING_MACHINE(0.3)
CAFE(0.8)
WASTE_BASKET(0.2)
CLOCK(0.3)
BENCH(0.2)
CAR_WASH(0.8)
CAR_RENTAL(1.0)
BAR(0.6)
ICE_CREAM(0.4)
PHARMACY(1.2)
CLINIC(1.0)
FIRE_STATION(1.8)
VETERINARY(1.0)
SHELTER(0.6)
CHILDCARE(1.1)
DENTIST(1.3)
NURSING_HOME(1.4)
ARTS_CENTRE(1.1)
DOCTORS(1.5)
BROTHEL(0.5)
PREP_SCHOOL(1.3)
THEATRE(1.5)
COMMUNITY_CENTRE(1.2)
COLLEGE(1.4)
SOCIAL_FACILITY(1.1)
TOWNHALL(1.3)
NIGHTCLUB(0.8)
DRINKING_WATER(0.4)
BUREAU_DE_CHANGE(0.9)
CHARGING_STATION(1.0)
FOOD_COURT(0.6)

TAXI(0.5)
HOSPITAL(2.0)
CASINO(0.7)
PHOTO_BOOTH(0.3)
BICYCLE_RENTAL(0.5)
WATER_POINT(0.2)
SANITARY_DUMP_STATION(0.4)
TOILETS(0.3)
SHOWER(0.5)
BINOCULARS(0.4)
PUBLIC_BOOKCASE(0.4)
LUGGAGE_LOCKER(0.5)
PARKING_SPACE(0.5)
MARKETPLACE(0.7)
HUNTING_STAND(0.2)
SUPERMARKET(1.5)
CONVENIENCE(0.9)
CHEMIST(1.0)
ALCOHOL(0.5)
BAKERY(0.7)
BUTCHER(0.8)
GIFT(0.6)
FLORIST(0.7)
OUTDOOR(0.8)
SOFT_DRUGS(0.5)
BICYCLE(0.6)
ELECTRONICS(1.0)
SPORTS(0.9)
CHARITY(0.4)
OPTICIAN(0.7)
HEARING_AIDS(0.6)
SHOES(0.7)
CLOTHES(1.0)
ICE(0.3)
STATIONERY(0.5)
FURNITURE(1.0)
CURTAIN(0.4)
KITCHEN(1.0)
HAIRDRESSER(1.2)
VACANT(0.1)
CAR_REPAIR(0.8)
CAR(1.0)
GARDEN_CENTRE(0.7)
LAUNDRY(0.5)
COMPUTER(1.0)
DEPARTMENT_STORE(1.4)
COFFEE(0.5)
MUSIC(0.8)
TOYS(0.6)
NEWSAGENT(0.4)
DOITYOURSELF(1.0)
GREENGROCER(0.8)
WINE(0.7)
BOOKS(0.9)
MEDICAL_SUPPLY(1.1)
INTERIOR_DECORATION(0.8)
PASTRY(0.6)
PARTY(0.4)

BEAUTY(0.9)
NUTRITION_SUPPLEMENTS(0.6)
DRY_CLEANING(0.5)
MOBILE_PHONE(1.0)
JEWELRY(1.2)
CONFECTIONERY(0.4)
DELI(0.6)
COSMETICS(1.0)
PET(0.5)
PHOTO(0.4)
ACCESSORIES(0.6)
ANTIQUES(0.7)
PERFUMERY(1.0)
CHOCOLATE(0.3)
HOUSEWARE(0.7)
ART(1.1)
BAG(0.5)
TOBACCO(0.6)
VARIETY_STORE(0.7)
WATCHES(0.8)
EROTIC(0.5)
CANNABIS(0.5)
DRUGS_PARAPHERNALIA(0.4)
PAINT(0.3)
BEVERAGES(0.6)
HIFI(0.8)
SECOND_HAND(0.5)
STORAGE_RENTAL(0.4)
TICKET(0.4)
WHOLESALE(0.9)
TEA(0.4)
KIOSK(0.5)
CAR_PARTS(0.8)
PAWNBROKER(0.3)
BABY_GOODS(0.6)
MARKET(0.7)
CRAFT(0.5)
CARPET(0.6)
BED(0.8)
MALL(1.2)
MASSAGE(0.7)
HAIRDRESSER_SUPPLY(0.5)
FABRIC(0.4)
MUSEUM(1.5)
VIEWPOINT(0.8)
ARTWORK(0.9)
HOTEL(1.7)
ATTRACTION(1.4)
ZOO(1.3)
HOSTEL(1.1)
GUEST_HOUSE(1.2)
APARTMENT(1.0)
CARAVAN_SITE(0.6)
GALLERY(1.3)