Université de Mons
Faculté Polytechnique

Route-Finding System for Points in Wallonia

Geospatial Data Management and Analysis. Principles and applications - I-GEOL-200

Students

Rafael Menas Tamasi	Université de Mons, Belgium
Bruno Ruiz Juarez	Université de Mons, Belgium
Aurélien Niebes	Université de Mons, Belgium

Professor

Dr. Olivier Kaufmann	Université de Mons, Belgium

Mons, Wallonia
Dec/06/2024

Abstract

This project focuses on developing a Python application within JupyterLab to calculate optimal routes between two points using bus or walking paths. By getting datasets from OpenStreetMap, GTFS files from the TEC (the wallonian bus operator) and libraries such as `osmnx`, `folium`, and `networkx`, the project intends to give to users a fast route. The process involves generating isochrone maps and applying Dijkstra's algorithm to analyze and refine routes based on nearby bus stops.

Introduction

Route optimization plays a vital role in geospatial applications, offering solutions for improving navigation and accessibility. This project aims to address the need for efficient travel planning by developing a Python application within JupyterLab that calculates optimal routes between two points. The tool leverages publicly available OpenStreetMap data and GTFS files from the bus operator, ensuring accessibility and adaptability for various use cases.

By integrating powerful libraries such as `osmnx`, `folium`, and `networkx`, the application supports the calculation of fast and efficient walking and bus routes. The project also employs isochrone map generation and Dijkstra's algorithm to refine results iteratively, ensuring comprehensive route analysis.

This report outlines the methodology, implementation, and results of the project, emphasizing the value of geospatial tools in modern urban planning and transport optimization. It further demonstrates the potential applications of this tool in academic and practical scenarios, contributing to the broader field of geospatial data management.

Literature Review

Route optimization has been extensively studied due to its importance in transportation and urban planning. Tools such as Google Maps and OpenStreetMap provide widely-used solutions, but these often rely on proprietary algorithms and interfaces. In contrast, open-source libraries like `osmnx` and `networkx` enable researchers to implement customizable solutions tailored to specific applications.

Dijkstra's Algorithm: A Key Approach to Route Optimization

Dijkstra's algorithm, introduced by Edsger Dijkstra in 1956, is a fundamental method for finding the shortest path between nodes in a graph.

How Dijkstra's Algorithm Works

The algorithm operates on a weighted graph, where nodes generally represent locations, and weighted edges can represent paths with associated costs (e.g. distance, time), fitting perfectly into the context of the project, the way we implemented this alorithm was creating a graph where the nodes represented the stations and the edges where weighted by the time it takes to go from one station to the other, either walking or via bus. The goal is to determine the shortest path from a source node to all other nodes in the graph. As shown in Figure 1, the key steps of the algorithm are as follows:

1. Initialization:
    - Assign an initial distance of infinity (‘∞‘) to all nodes except the source node, which is set to 0.
    - Mark all nodes as unvisited and create an empty set for visited nodes.
2. Selection of the Current Node:
    - Select the unvisited node with the smallest distance value. This node is considered the current node.
3. Relaxation of Neighbors:
    - For each neighboring node of the current node, calculate the tentative distance by adding the current node's distance to the edge weight.

- If the tentative distance is smaller than the previously recorded distance, update it.

4. Mark as Visited:
    - Once all neighbors of the current node are processed, mark the current node as visited. A visited node will not be checked again.
5. Repeat Until Completion:
    - Repeat steps 2–4 until all nodes are visited or the target node's shortest distance is finalized.

Conclusion about Dijkstra

Dijkstra's algorithm is used in multiple fields thanks to its simplicity and efficiency. its intuitive to use dijkstra in this kinds of problem due that our application fits perfectly into a transport problem with one initial node and one final node, allowing us to implement easily the isochrone maps that allow us to take decisions not only in the spatial aspect of the problem instead tackling it via the time aspect

Overview of Existing Tools and Methods for Route Optimization

Route optimization is a critical component of modern navigation systems; as already mentioned many branches of science and technology have tacked transport problems, so luckily we counted with a variety of tools and methods developed to address the challenges of transport planning. This section provides an overview of widely used solutions such as BBBike and OpenStreetMap, highlighting their features, methodologies, and relevance to the project.

OpenStreetMap: A Collaborative Open-Source Platform

OpenStreetMap (OSM) is an open-source, community-driven project that provides geospatial data for a wide range of applications. It serves as a foundation for route optimization in various tools. Notable features of OSM include:

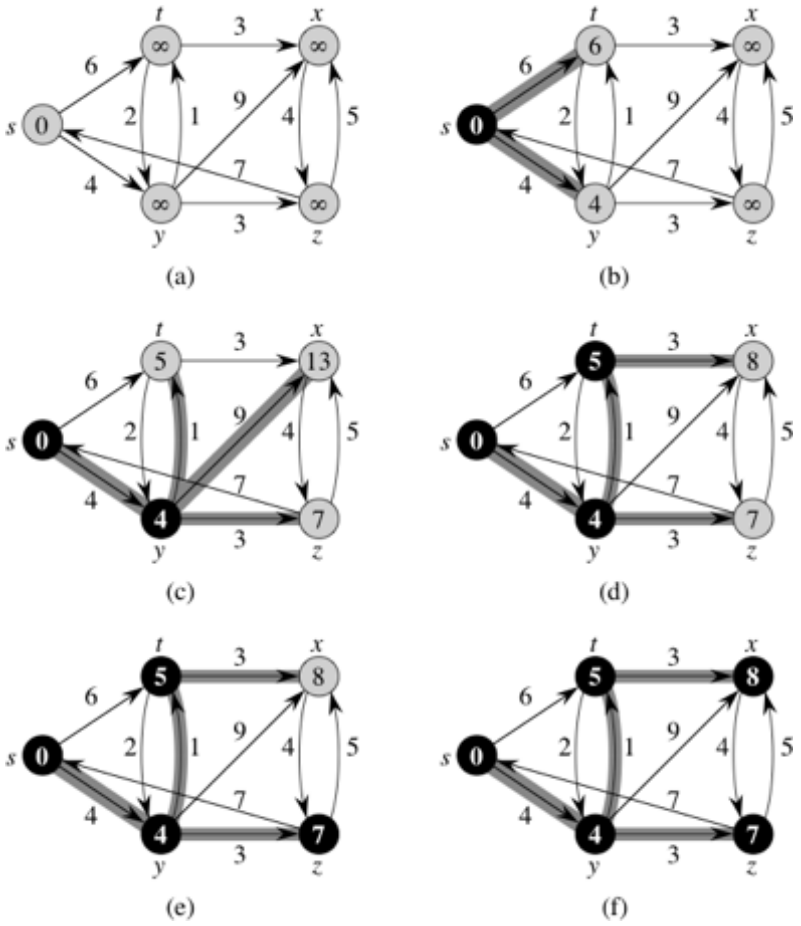

Figure 1: Dijkstra Algorithm working.
https://www.cs.dartmouth.edu/ thc/cs10/lectures/0509/0509.html.

- **Collaborative Data Contributions:**
  - Users contribute and update data on roads, landmarks, and transit routes.
  - Community efforts ensure up-to-date and detailed local mapping.
- **Customizability:**
  - Open-source nature allows developers to tailor data for specific projects.
  - Enables integration with libraries like `osmnx` and `networkx`.

- **Global Coverage:**
  - Extensive geographical coverage, especially in areas underserved by proprietary solutions.
  - Supports a wide range of applications, from humanitarian efforts to academic research.

For this project, OpenStreetMap is chosen due to its compatibility with Python libraries such as `osmnx` and its ability to integrate seamlessly with iterative optimization techniques.

Conclusion about the tools

The comparison of these tools illustrates the trade-offs between proprietary and open-source solutions in route optimization. By leveraging OpenStreetMap's open data and modularity, this project benefits from a flexible and accessible platform for geospatial analysis.

Overview of Libraries Used

This project relies on four key Python libraries—`osmnx`, `folium`, `networkx` and `GeoPandas`—to enable route optimization and geospatial analysis. Each library provides functionalities, making them good tools for the implementation of the application.

`osmnx`: Simplifying OpenStreetMap Data Access

`osmnx` is a powerful library designed for working with OpenStreetMap data, providing tools for geospatial analysis and graph-based operations. Key features include:

- **Graph Creation:**
  - Automatically downloads and converts OpenStreetMap data into graph structures.
  - Supports both directed and undirected graphs for flexible route analysis.
- **Custom Queries:**

  - Allows users to filter data based on tags such as roads, amenities, and transit stops.
  - Enables customization to specific geographic boundaries or route preferences.

- **Visualization:**
  - Provides tools to plot street networks and routes directly on maps.
  - Compatible with other libraries for advanced visualizations.

`osmnx` is essential for retrieving and processing OpenStreetMap data, forming the backbone of the project's geospatial operations.

`folium`: Interactive Map Visualization

`folium` is a library designed for creating interactive maps, making it ideal for presenting geospatial data dynamically. Its key features are:

- **Map Customization:**
  - Supports the integration of tilesets such as OpenStreetMap, Stamen, and Mapbox.
  - Enables users to overlay routes, markers, and isochrone areas.
- **Interactivity:**
  - Allows zooming, panning, and toggling of map layers.
  - Provides popups and tooltips for additional data visualization.
- **Exportability:**
  - Generates HTML files, making maps shareable and accessible across platforms.

By integrating `folium`, the project enhances user experience with dynamic and interactive visualizations of routes and isochrone maps.

`networkx`: Graph-Based Analysis

`networkx` is a versatile library for creating, analyzing, and manipulating graph structures, central to route optimization. Its core features include:

- **Graph Operations:**

  - Facilitates the creation and manipulation of graph structures.
  - Supports weighted edges, critical for representing travel distances or times.

- **Algorithm Implementation:**
  - Includes a wide range of algorithms, such as Dijkstra's algorithm, for shortest path computation.
  - Supports clustering, connectivity, and centrality analysis.

- **Compatibility:**
  - Easily integrates with data from `osmnx`, allowing for seamless graph creation and analysis.

`geopandas`: Spatial data sets manipulation

`networkx` is integral to implementing route optimization algorithms, ensuring efficient pathfinding and data manipulation.

`GeoPandas` is a library enabling the manipulation of huge datasets of geographical and non-geographical data. It is built upon the libraries `Pandas` and `shapely` from which it takes the dataset manipulation function and the geometric operations. Its key features are :

- **Geo-spatial datasets:**
  - Extends `Pandas`'s dataframe with "geodataframe" which include geoseries columns storing `shapely` geometry objects and series storing "traditional" data (number, strings, boolean, etc.).
  - Stores information about the geodataframe projection in its CRS attribute (CRS stands for Coordinate Reference System). Different geoseries of a geodataframe may have different crs.

- **Geometrical operations:**
  - Apply geometrical operations (including re-projection) to all the objects in a geoseries at once, which reduces code complexity and overhead.
  - Allow spatial filtering of geodataframes based on relations with other spatial objects.

- **Reading and Writing files**

  - Permit to process common geospatial files (e.g. GeoPackage, GeoJSON, ...) and convert them into geodataframes.
  - Allow geodataframes to be stored in multiple geospatial files.

- **Geodatasets visualization:**
  - Generates static or interactive map of the geodatasets using `matplotlib` or`folium`

`Geopandas` is crucial for retrieving and handling GTFS files alongside other geospatial datasets.

Relevance to the Project

The combination of these three libraries enables the project to perform end-to-end geospatial analysis:

- `osmnx` handles data retrieval from OpenStreetMap and graph preparation.
- `folium` provides interactive visualizations for better user engagement.
- `networkx` implements algorithms and handles graph computations.
- `geopandas` facilitates the extraction of data from the GTFS files and the manipulation of the geospatial datasets.

Conclusion

The synergy between `osmnx`, `folium`, `networkx` and `geopandas` ensures a seamless workflow for this project, from data acquisition to analysis and visualization.

Together, these tools form a robust framework for optimizing and visualizing routes, demonstrating the power of open-source software in geospatial applications.

5. Methodology

The methodology employed in this project involves developing a Python application for geospatial route optimization. These steps cover data collection, application design, route calculation, isochrone map generation, bus stop location, and optimization algorithms.

5.1 Data Collection

Data collection is a foundational step that ensures accurate and relevant inputs for the application. The primary data sources are OpenStreetMap (OSM), which provides information on walking paths and bus stops, and the GTFS files provided by the TEC, Wallonia's bus operator, which contains the schedule of the bus trips, complementary information about the bus stops, and the paths taken by the buses.

The data from OpenStreetMap are processed as follows: walking paths are extracted as graph edges that represent pedestrian routes, while bus stop data are collected and mapped as nodes for analysis within isochrone areas.

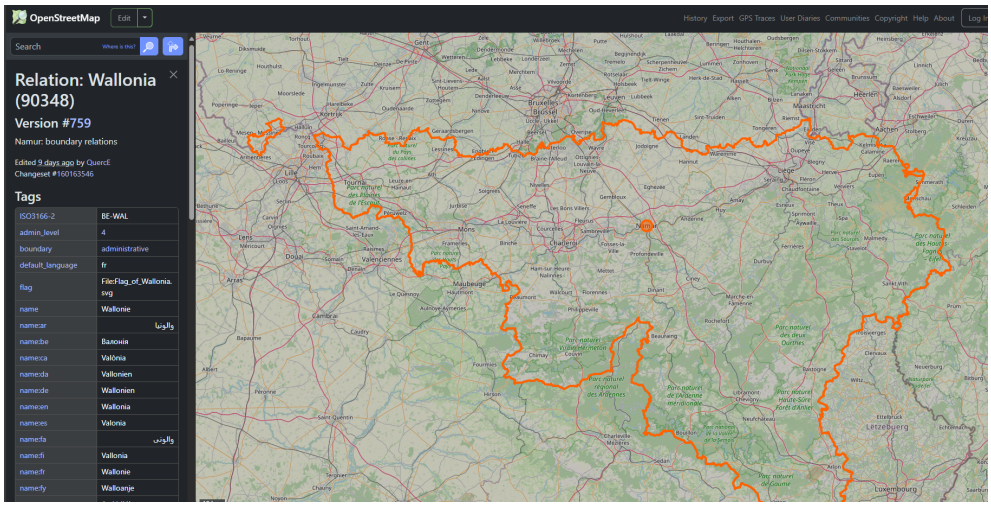

Figure 2: OpenStreetMap wallonia zone

On the other hand, the data from each GTFS file is extracted and stored individually in dataframes. To save on memory usage, we don't import empty columns and use more efficient data types when possible. Since the arrival and departure times of the buses can be above 24:00, we convert them into

TimeDelta. Afterwards, the dataframes containing the bus stops and the paths must be further preprocessed to convert them into geodataframes. Regarding the bus stops, their positions are stored as coordinates that must be converted into spatial points. For the paths, they are stored as coordinates that follow each other in a given order. So, first, they are converted into points, which are then grouped into lines.

5.2 Application Design

The Python application is developed in JupyterLab and integrates libraries for geospatial and graph-based analysis, its design includes modular code blocks that facilitate data processing, analysis, and visualization. User inputs are managed through code commands to provide flexibility.

5.3 Identification of active bus trips

The bus schedule is different depending on the days, for example, if it is the weekend of the working week. The GTFS files account for this by linking the bus stops to "services". These services represent the days of the week and the time span on which given bus trips are active. So, before the algorithm starts, we find the day of the departure date and the associated services. From this we can find the trips and the stop times that are active.

Some schedule exception can be stored in a separate file called "calendar_dates.txt" to, for example, indicate that a bus trip is not active on Christmas day. For the moment, our algorithm ignore these exception but the could be implemented in the future. Our algorithm assumes that only the bus trips on the same day that the departure should be considered.

5.4 Route Calculation

The application calculates optimal routes between two user-defined points. For the moment, those points should be bus stops. However, with some changes in the code, users could provide the coordinates of these points through code commands in JupyterLab, and the application could process this input to define the start and end nodes in the graph.

The shortest path is found by using a custom variation of Dijkstra's algorithm. Indeed, because the bus trips available between stops variate depending of time, we can't generate a bus trips network to solve the problem for any time. Instead, we explore the bus trips available at the given departure time and date while running the algorithm. It is similar to constructing the algo-

rithm as we explore it. This algorithm starts from a departure bus stop which name is given by the user and find all the bus stops directly linked to it by a trip or in walkable distance (whe chose arbitrary 1km the maximum walkable distance). The arrival time to these stops is either retrieved from the schedule in the GTFS files or by computing the shortest walking time between stops. If several bus trips exist between two bus stops (as it is often the case), only the trip with the earliest arrival time is kept.

The number of paths to go to one stop is stored and if it is higher than a given number, when exploring this bus stop, instead of finding the bus stops connected to it, the function return an empty dataset, as if the bus stop was connected to nothing. This limit the number of transfers of the final best path and the computation time at the cost of finding a sub-optimal path if the departure and the arrival are too far.

To compute the walking time between two bus stops, the algorithm uses osmnx to create a graph representation of the area based on OpenStreetMap data. Then, networkx is employed to compute the shortest walking route using Dijkstra's algorithm (which amounts to solving a smaller shortest path problem).

For all these stops, the distance to the arrival point is computed and divided by a walking speed to estimate the walking time to the arrival point. This walking time is added to the arrival time to estimate the final arrival time to the arrival. We choose to use only the bird-fly distance and not the exact walk paths to save time. If a estimated final arrival time is earlier than the best one yet, all the bus trips that arrive later than it at a stop are deleted to limit the number of stops to explore.

All the stops connected to the departure stop are then appended to the dataframe of the known stops and the departure stop is marked as explored. The dataframe of the known stops is sorted by the estimated final arrival time and the first unexplored stop is chosen to be explored. This loop continues until all the stops are explored.

5.5 Isochrone Map Generation

Isochrone maps are used to visualize areas accessible within specific timeframes, their purpose is to help identify bus stops within a walkable range and provide visual feedback to validate route calculations. The generation of these maps involves using folium to create interactive isochrone maps, walking distances from the start point are computed, and regions are marked to indicate accessibility.

Results

The application successfully calculates optimal routes between two points using bus and walking paths. Key results include:

- **Walking Route Visualization**: Generated maps accurately display walking paths between selected points.

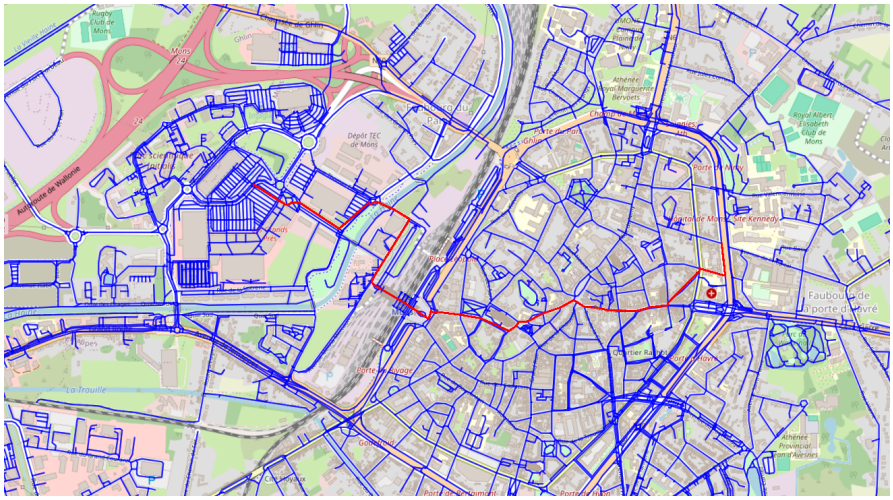

Figure 3: These are all the walkable paths of Mons extracted from OSM

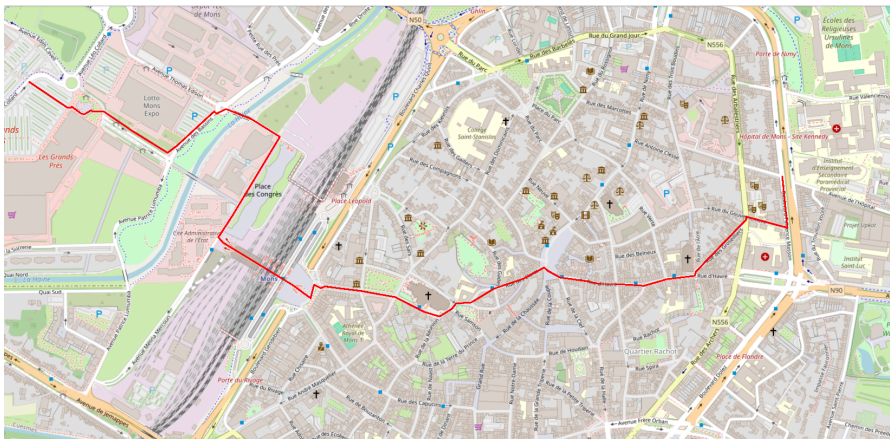

Figure 4: After filtering the walkable paths, we got the fastest one between the destination

- **Isochrone Maps**: Highlight areas reachable within defined timeframes, effectively integrating walking and bus stops.
- **Refined Routes**: Iterative use of Dijkstra's algorithm improves route efficiency by incorporating bus stops.
- **Performance**: The application demonstrates consistent accuracy in route calculations, with processing times remaining within acceptable limits.

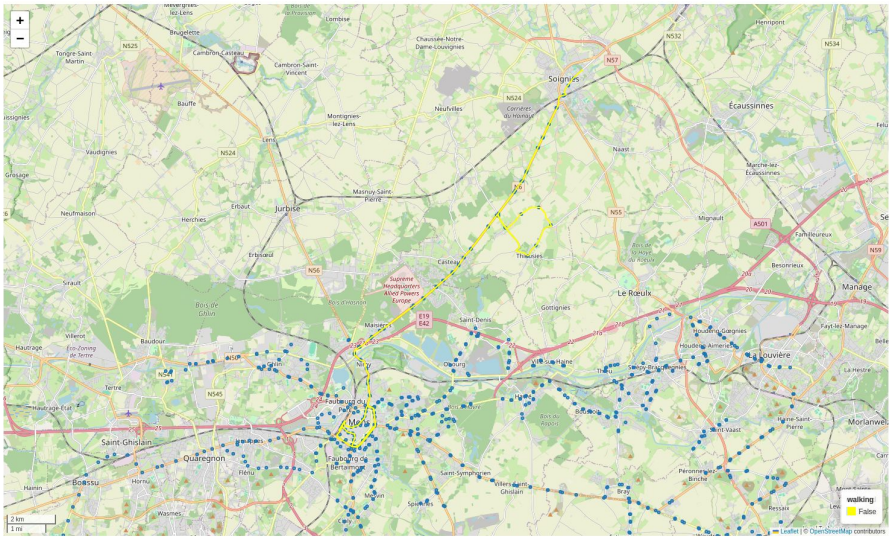

Figure 5: Fastest route plot

These results showcase the capability of the application to address real-world geospatial problems and provide practical navigation solutions.

Discussion

Even though the results meets the goals and expectations we identified areas of impovement, Strengths and challenges we met during the project development.

- **Strengths**: Modular design thanks to jupyther lab makes colaboration and code intengration fast and easy to keep track of, The use of an extensive search grantees the fastest way available with the data provided.
- **Challenges**: Computational complexity increases with smaller isochrone maps and dense urban areas, impacting performance.
- **Applications**: Beyond navigation, this tool can support urban planning, disaster response, and public transit optimization.
- **Future Improvements**: Adding a graphical user interface (GUI) and expanding data integration to include live transit updates could enhance usability and functionality.

Conclusion

This project successfully demonstrates the use of Python and open-source geospatial libraries for route optimization. By calculating walking routes, generating isochrone maps, and applying iterative analysis with Dijkstra's algorithm, the application provides a practical solution for navigation challenges.

The project also provided an opportunity to acquire knowledge in technologies and research fields that were new to most team members, including:

- CRS (Common Reference Systems)
- Projection
- Geodatabases
- Isochrone Concept
- GTFS (General Transit Feed Specification)

The results showcase the progress of the authors in these technologies and present potential for broader applications in geospatial analysis and urban planning. Future work could focus on improving user interaction, incorporating real-time transit data, other transport networks, and exploring additional optimization techniques.

In conclusion, the goals of learning about geospatial data management and its diversity of applications were met, as well as the goals determined by the work team to deliver a working application that gives a solution to the problem presented.

W

References

- Ahuja, R. K., Magnanti, T. L., Orlin, J. B. (1993). Network flow: Theory, algorithms, and applications. Prentice Hall.
- Javaid, M. A. (2013). Understanding Dijkstra's algorithm. SSRN. https://doi.org/10.2139/ssrn.2340905
- Hagberg, A. A., Schult, D. A., Swart, P. J. (2008). NetworkX: Network analysis with Python. Retrieved from https://networkx.github.io/
- Jordahl, K. (2020). GeoPandas: A Python package for geospatial data analysis. Retrieved from https://geopandas.org/
- OpenStreetMap Contributors. (n.d.). *OpenStreetMap*. Retrieved December 20, 2024, from https://www.openstreetmap.org
- BBBike. (n.d.). BBBike route planner. Retrieved December 20, 2024, from https://bbbike.org/
- Geoapify. (n.d.). Isochrone maps. Retrieved December 20, 2024, from https://www.geoapify.com/isochrone-maps/
- MobilityData. (n.d.). Référence—General Transit Feed Specification. Retrieved December 19, 2024, from https://gtfs.org/fr/documentation/schedule/reference/