VoroPlan: Unveiling Urban Mobility Gaps with Voronoi Diagrams

Introduction 2
Voronoi Diagrams 3
How to execute this project 4
VoroPlan Algorithm 5
    Seed points and datasets 5
    Geometric Constraints 7
    Implemented Algorithm 9
Interactive planning of new stations 12
Drawbacks 13
Conclusion 14
References 14

Author: Caio Valente (https://github.com/valentecaio/voroplan)

Date: 2023-12-10

Introduction

This project was initiated during the "Computational Geometry" master's class at PUC-Rio in 2023, under the guidance of Professor Waldemar Celes. The development and execution of the project were carried out by the student Caio Valente.

The intricate nature of urban mobility demands a nuanced exploration, particularly in challenging landscapes such as Rio de Janeiro. The goal of this project is to develop a tool capable of dissecting the existing metro and bike-share networks, offering decision-making assistance in identifying mobility gaps - the areas underserved by these networks.

The chosen approach involves Voronoi Diagrams, due to their inherent ability to partition space based on proximity to key points, in this case, transportation stations. This computational geometry tool aligns seamlessly with the spatial considerations of urban planning, making it an apt choice for our exploration.

The developed tool is an interactive web application that draws on real network data from official sources like Metro Rio and Bike Rio. It constructs and renders a dynamic map, depicting stations and the Voronoi Diagram they collectively form. Users can actively manipulate stations - moving, deleting, and creating them - and observe real-time updates to the Voronoi Diagrams. This instant visual feedback aims to help discerning gaps in the network.

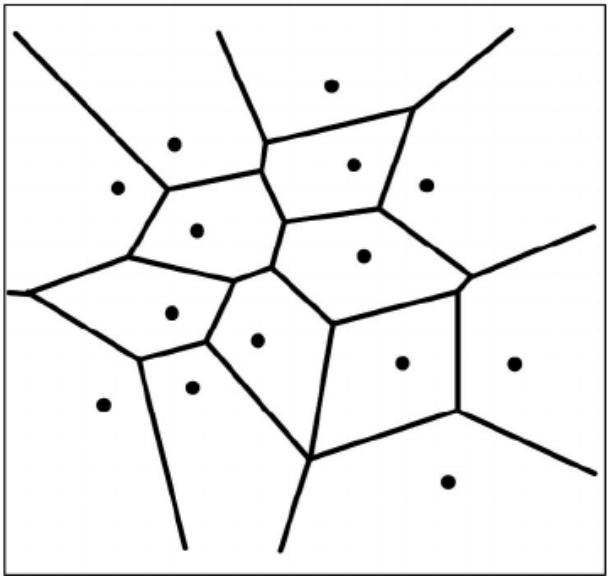
Additionally, the app allows the inclusion of obstacles in the diagram, allowing to exclude geographic obstacles - such as mountains and lagoons - from the diagram. The diagram cells are recalculated to contour these obstacles, aiming to provide valid mathematical comparison between different areas.

Voronoi Diagrams

Voronoi diagrams are a type of spatial tessellation that creates distinct polygons around a set of seed points. These polygons are called Voronoi cells, and they partition the space around the seed point into distinct regions. Each cell contains all the points in the space that are closer to its associated seed point than to any other seed point in the set.

In two dimensions, the Voronoi cell around a seed point is a polygonal region consisting of all the points closer to that point than to any other. The edges of the polygon are equidistant from the two nearest seed points.

Some edges of the Voronoi cells extend to infinity. These edges are formed by points that are equidistant from two or more seed points. For this reason, we can think of the Voronoi diagram as a partitioning of the plane into regions based on the distance to a specified set of points, with full coverage of the plane and no overlap between regions.



A Voronoi diagram

There are different metrics that can be used to define the distance between points. The most common is the Euclidean distance, which is the straight-line distance between two points. Other metrics include the Manhattan distance, which is the distance between two points measured along axes at right angles, and the Chebyshev distance, which is the maximum of the absolute difference of the coordinates of the points. The Voronoi diagram is defined by the distance metric used. In this project, we will use the Euclidean distance, although other metrics could be studied as well.

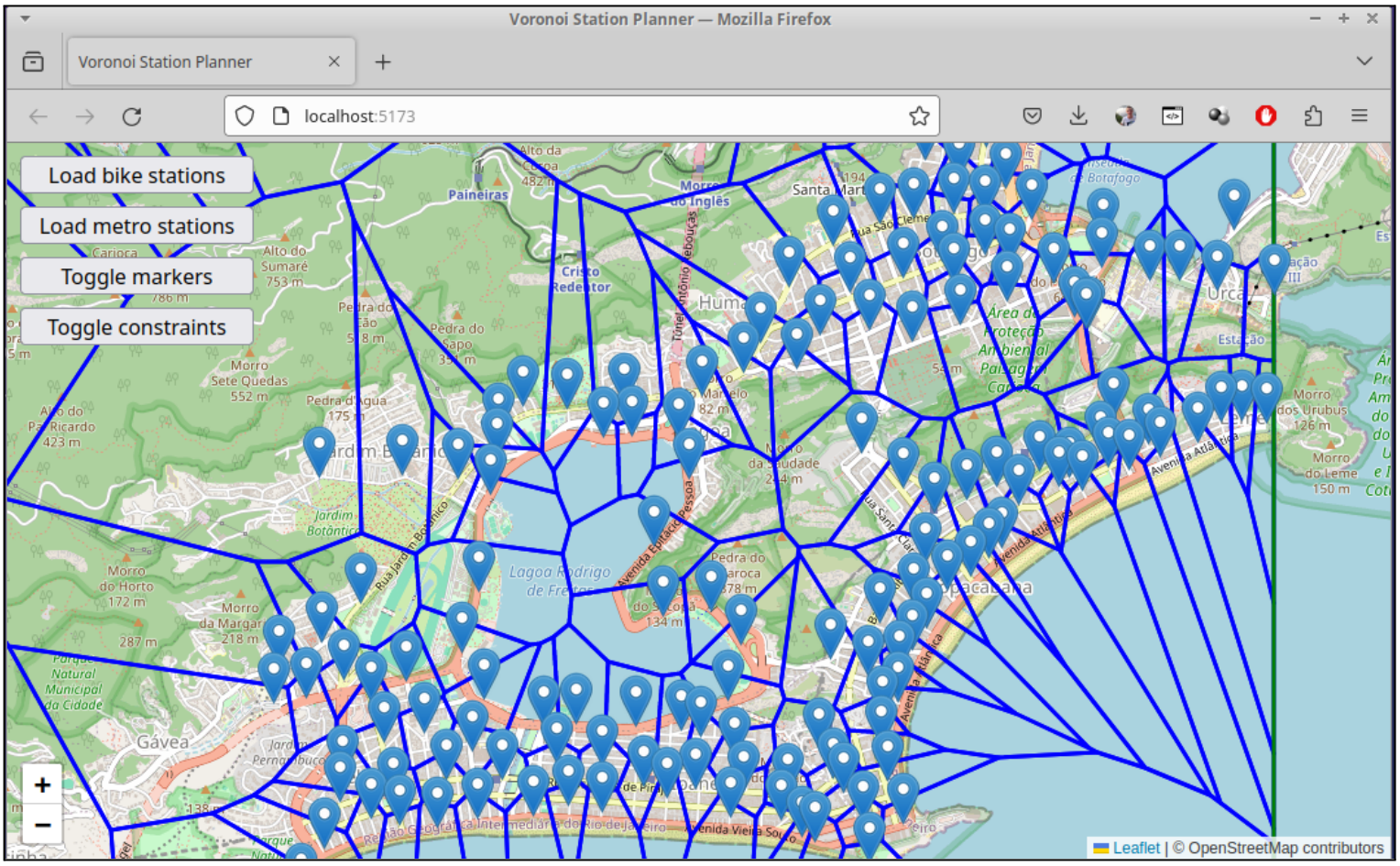
In our application, the seed points are the locations of the stations and the space is the area of the map that we want to cover. It can be a simple bounding box containing all the stations or a more complex polygon that contours the city more accurately. The Voronoi diagram will then partition the map into regions that are closer to a station than to any other station. This will allow us to determine which station is the closest to any given point on the map. The infinite edges will clip at their intersection with the outer polygon.

How to execute this project

To run the project locally and interact with the tool, follow the steps outlined below. The project is built using Node.js.

1. Install Node.js from nodejs.org.
2. Clone the repository from github.com/valentecaio/voroplan.
3. Install the dependencies:
   $ npm install
4. Run the node server:
   $ npm run dev
5. Access the application by visiting localhost:5173 in your browser.

The application should load a page containing a map and a voronoi diagram over it, such as the following example:



You can click on the map to create new stations, drag existing stations, or click on a marker to delete a station. New stations will be shown in red and original stations will change to orange if moved. Hovering on a marker will show a tip with the station name.

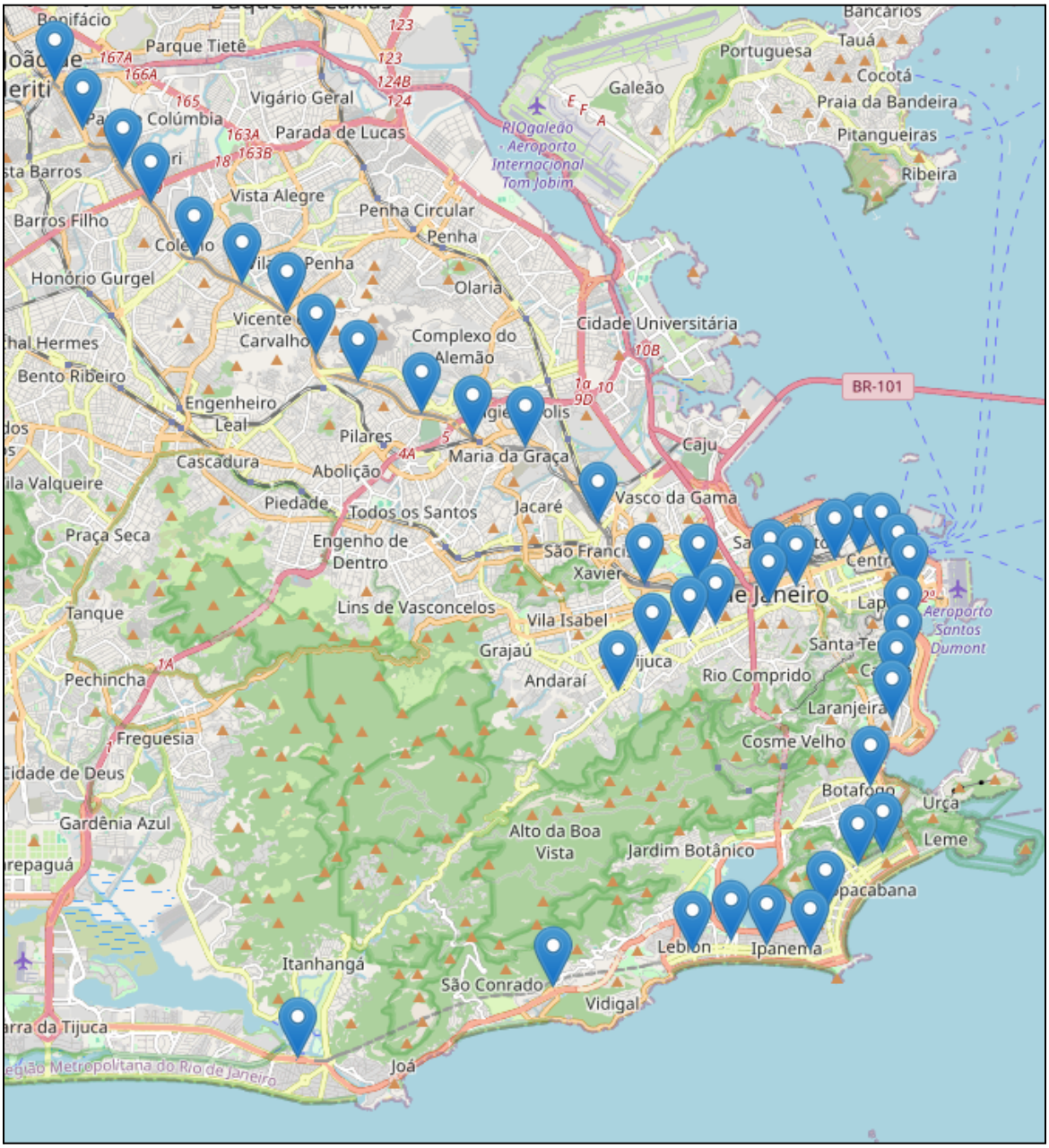
There are also some buttons on the left-top corner that allow changing datasets (bike/metro), hiding point markers and enabling/disabling constraints in the Voronoi diagram.

VoroPlan Algorithm

Seed points and datasets

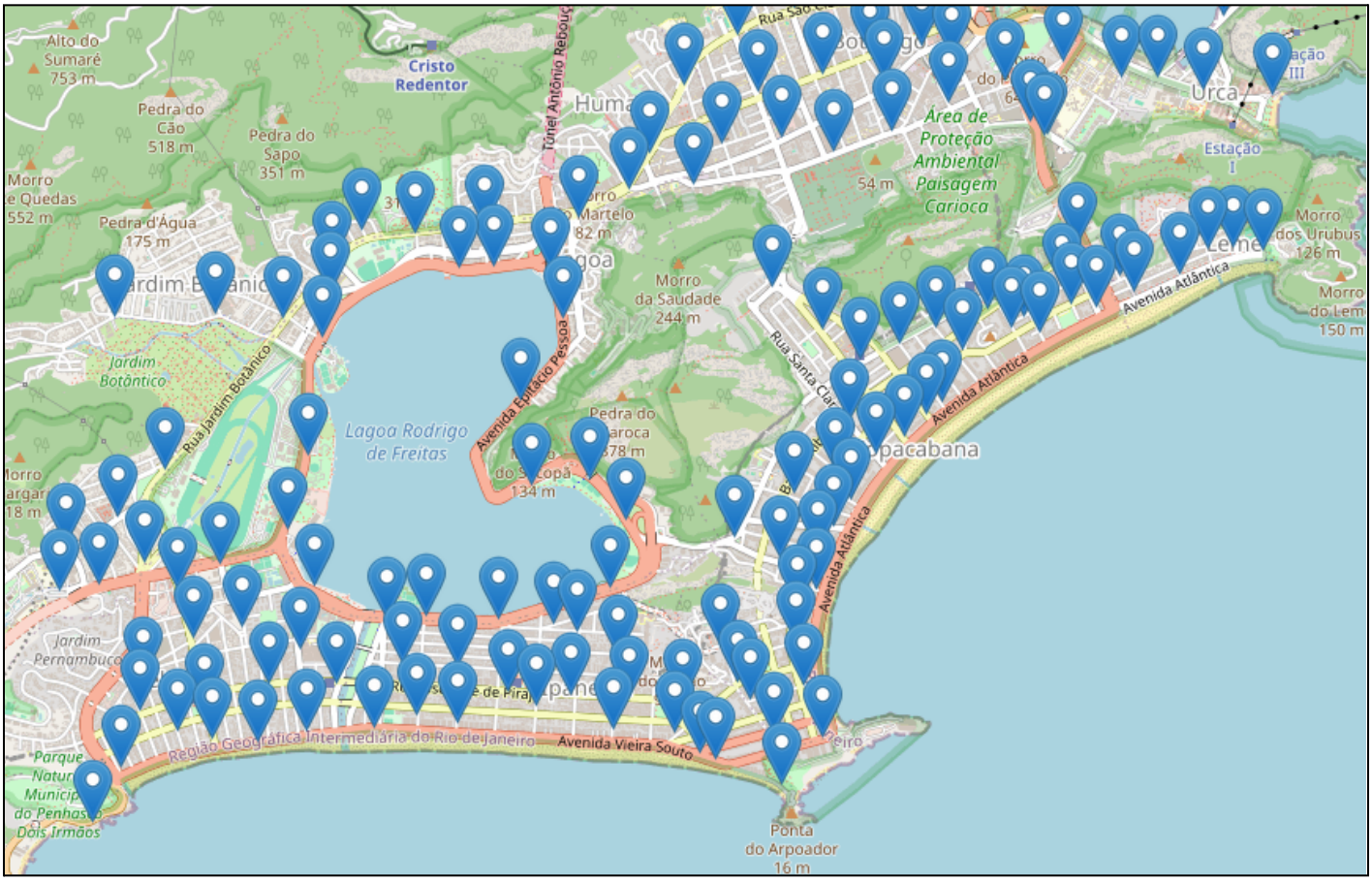
The datasets contain lists of station coordinates, where each station has a latitude, a longitude and a name. These points are the seed points, the input of our voronoi diagram.
I used two different input datasets:

Name	Number of stations	Obtained at
rio_bikes.csv	404	https://riodejaneiro.publicbikesystem.net/customer/ube/gbfs/v1/
rio_metro.json	41	https://www.data.rio/datasets/esta%C3%A7%C3%B5es-metr%C3%B4/explore



Metro Rio Stations dataset

Bike Rio Stations dataset



Bike Rio Stations dataset zoomed at the South Zone

Geometric Constraints

To start the algorithm, we define a bounding polygon that contains all the seed points and the area of the map that we want to cover with the network.

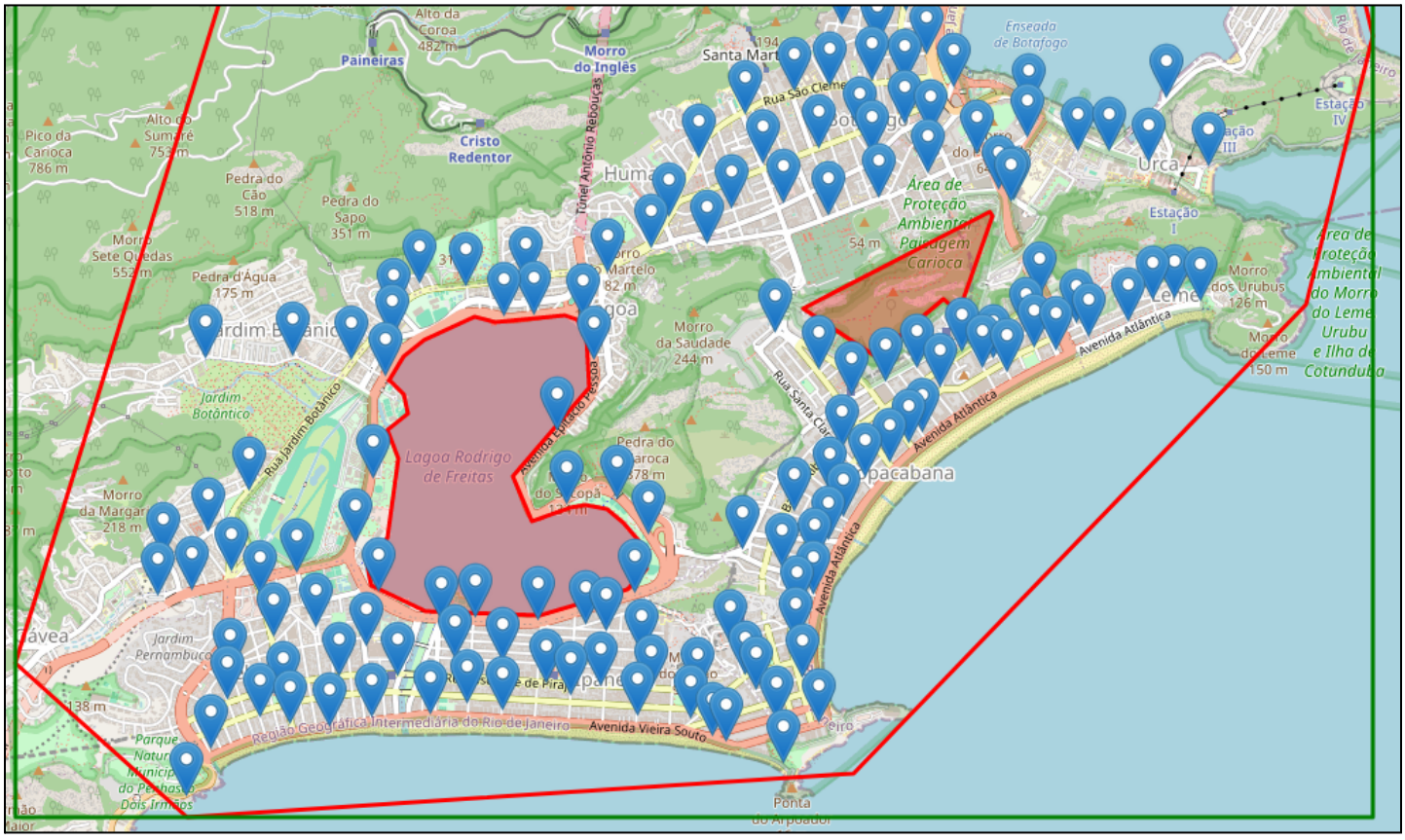
The simplest bounding polygon is a rectangle. However, this is not always the best choice, as it can lead to very elongated cells.

A better choice could be to use the convex hull of the seed points, which is the smallest convex polygon that contains all the seed points. This will lead to more compact cells. However, the convex hull may exclude some zones of the map that we could want to cover.

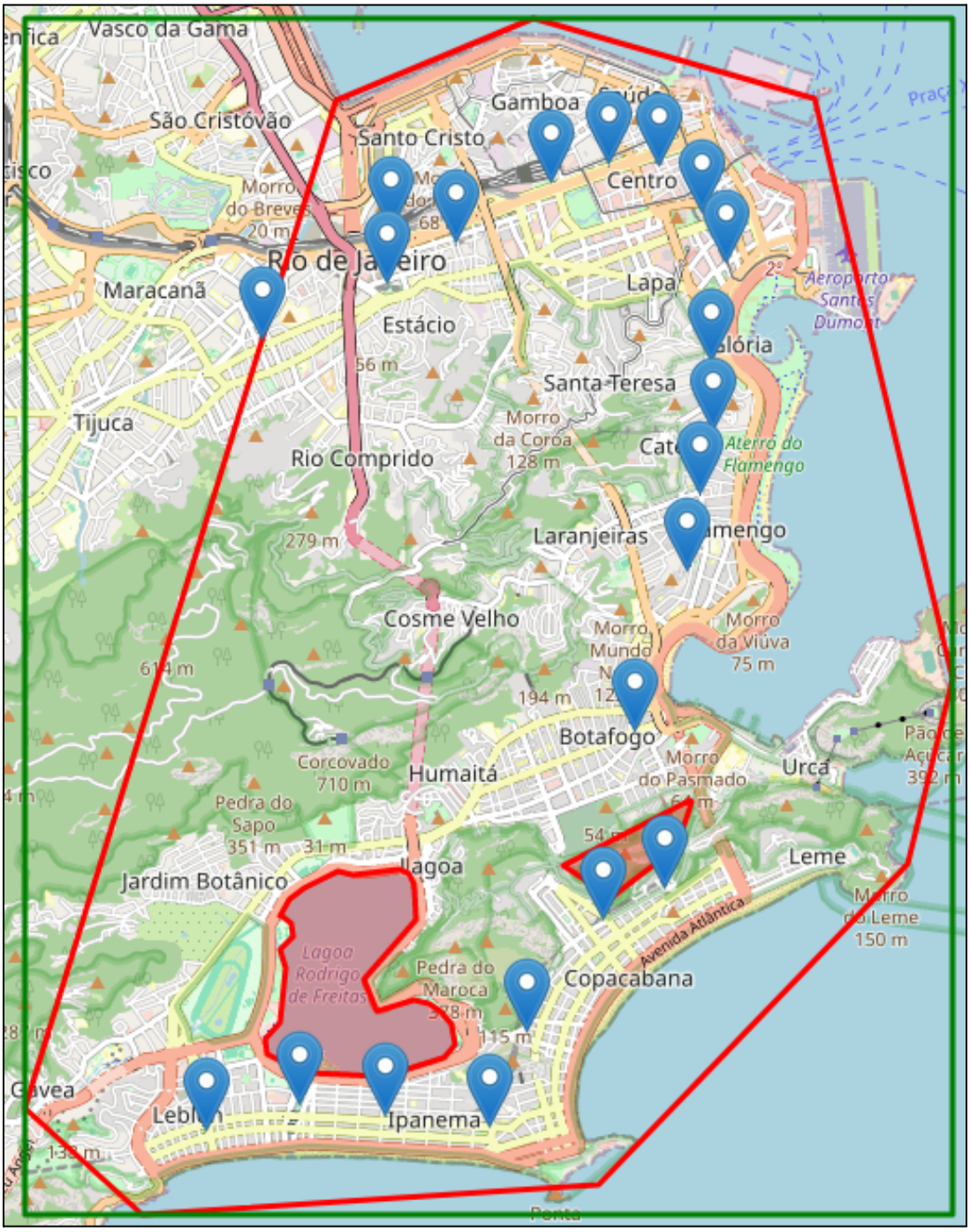
The best solution I could find was to manually define an outer convex polygon that contours the city and the areas of interest. This ensures that the Voronoi cells will cover the entire area of interest, and not just the area that contains the seed points. The outer polygon is defined in a json file (rio_constraints.json) that is loaded at the beginning of the algorithm. Additionally, as the datasets are too sparse, the outer polygon used in this study contains only the south and center zones of the city, where the majority of the stations of both datasets are located. The points outside the outer polygon are discarded from the diagram. This polygon must be convex by a limitation of my algorithm.

The same dataset also contains a list of inner boundaries, or obstacles, which are closed polygons inside our area of interest that represent parts of the map that we want to exclude from the diagram. These polygons could represent geographic zones that are inhabited and thus do not need to be served by the mobility network, such as mountains and lagoons. In the studied example, we have two inner polygons, the Rodrigo de Freitas Lagoon and the Paisagem Carioca Natural Park.

An additional tool could be added to the application to allow defining outer and inner polygons interactively.



Red lines compose the outer and inner polygons (bikes dataset).
Green lines are the bounding rectangle of the outer polygon.



Full view of the outer polygon (metro dataset)

Implemented Algorithm

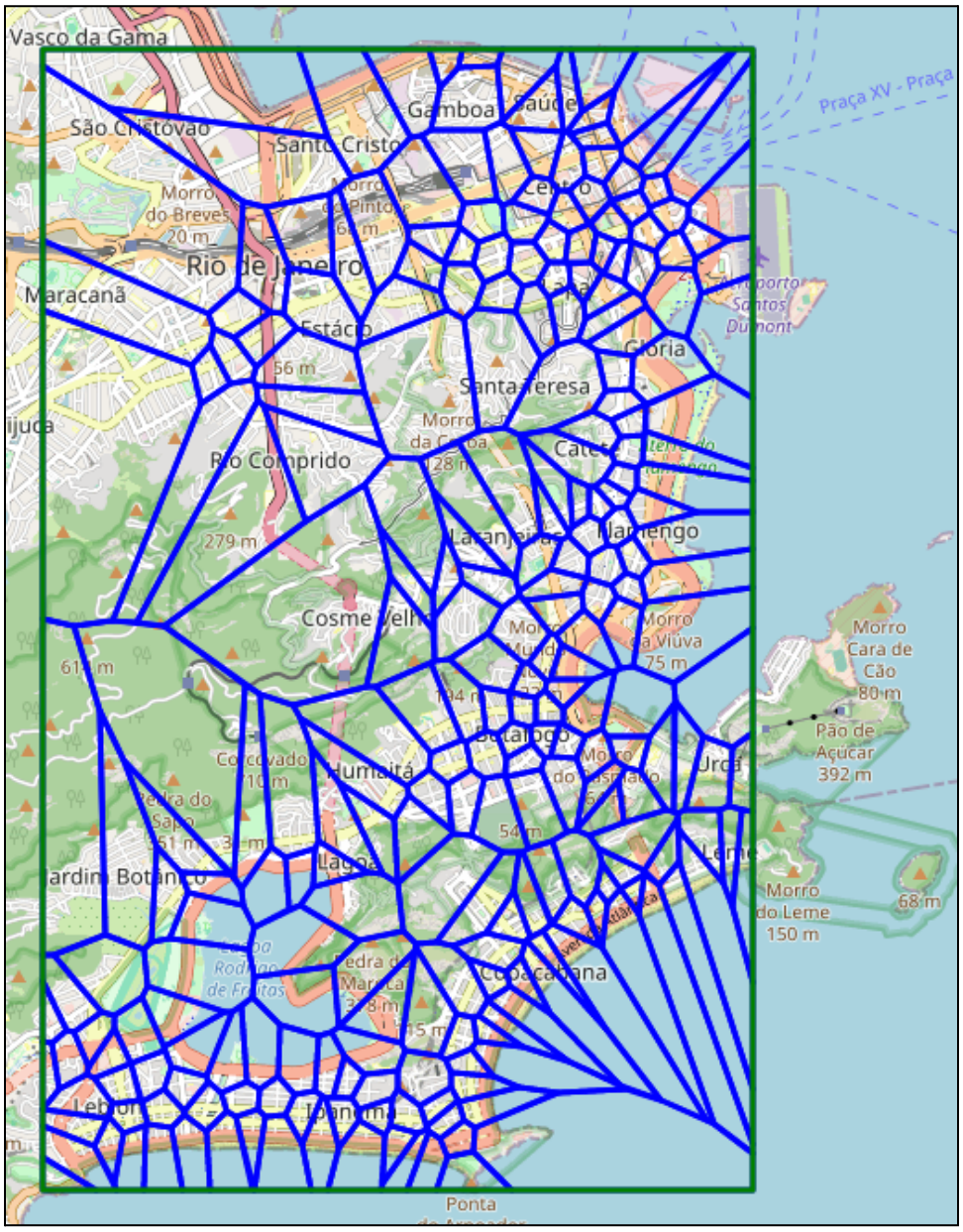
The problem of voronoi tessellation with obstacles is a bit more complicated. The simplest solution is to remove the obstacles from the bounding polygon and then compute the Voronoi diagram. However, this can lead to cells that are not connected to the rest of the diagram. A better solution is to use a constrained Delaunay triangulation, which is a triangulation of the seed points that respects the obstacles. The Voronoi diagram can then be computed from the constrained Delaunay triangulation. This ensures that the Voronoi cells will be connected to the rest of the diagram.

In this study, we adopted a simpler approach that has some limitations. We create a Voronoi diagram without obstacles, and then we add the obstacles, adapting the voronoi cells to contour them.

The main algorithm is implemented in the src/main.ts file and it works like the following:

1. Apply boundaries to seeds:
   The points that are out of the outer polygon or inside any obstacle cannot be part of our final Voronoi Diagram, so we discard them. This step only discards points that are out of our zone of interest.

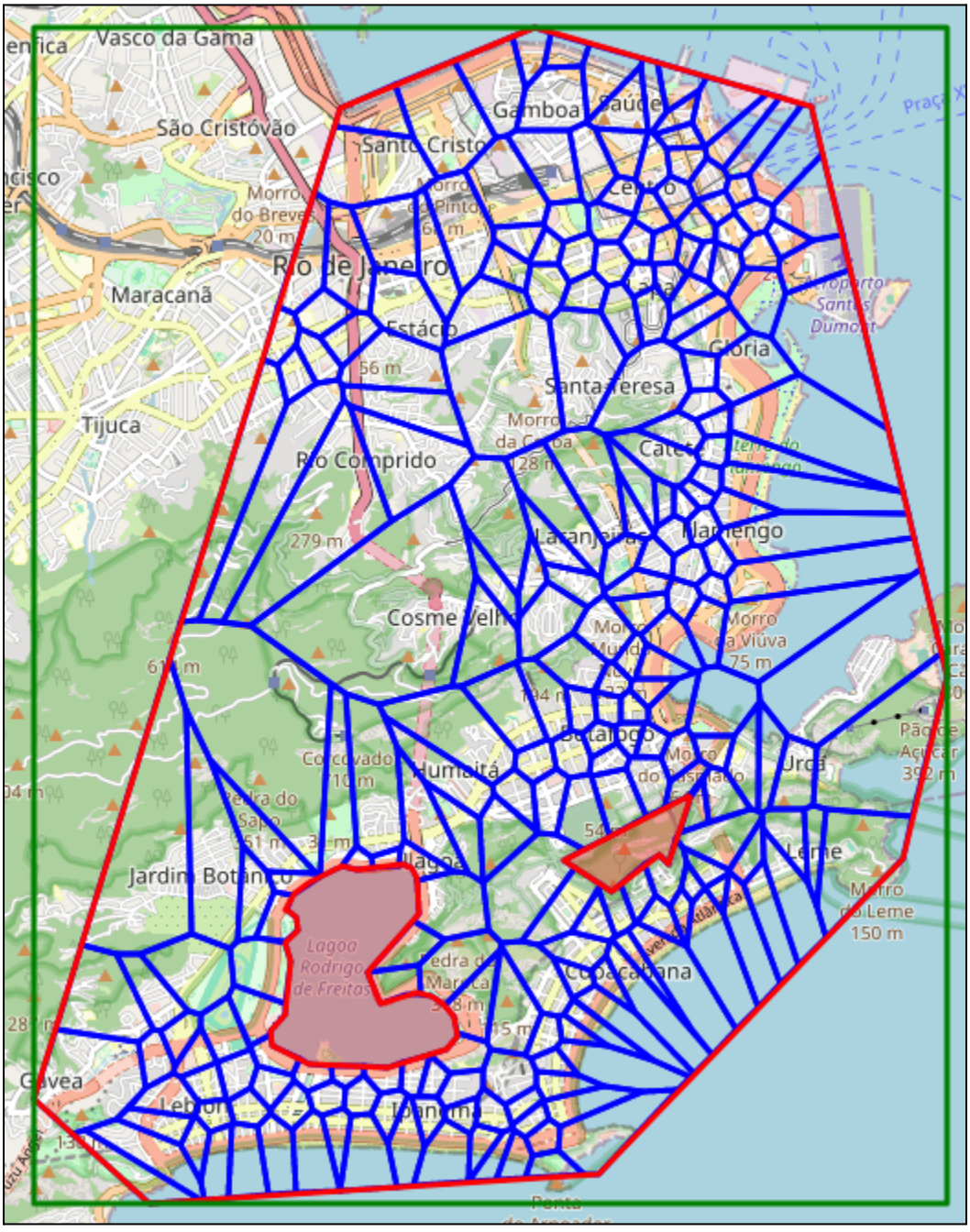
2. Create Voronoi Diagram without obstacles:
   The D3 library is used to compute a simple voronoi bounded by the bounding rectangle of the dataset.



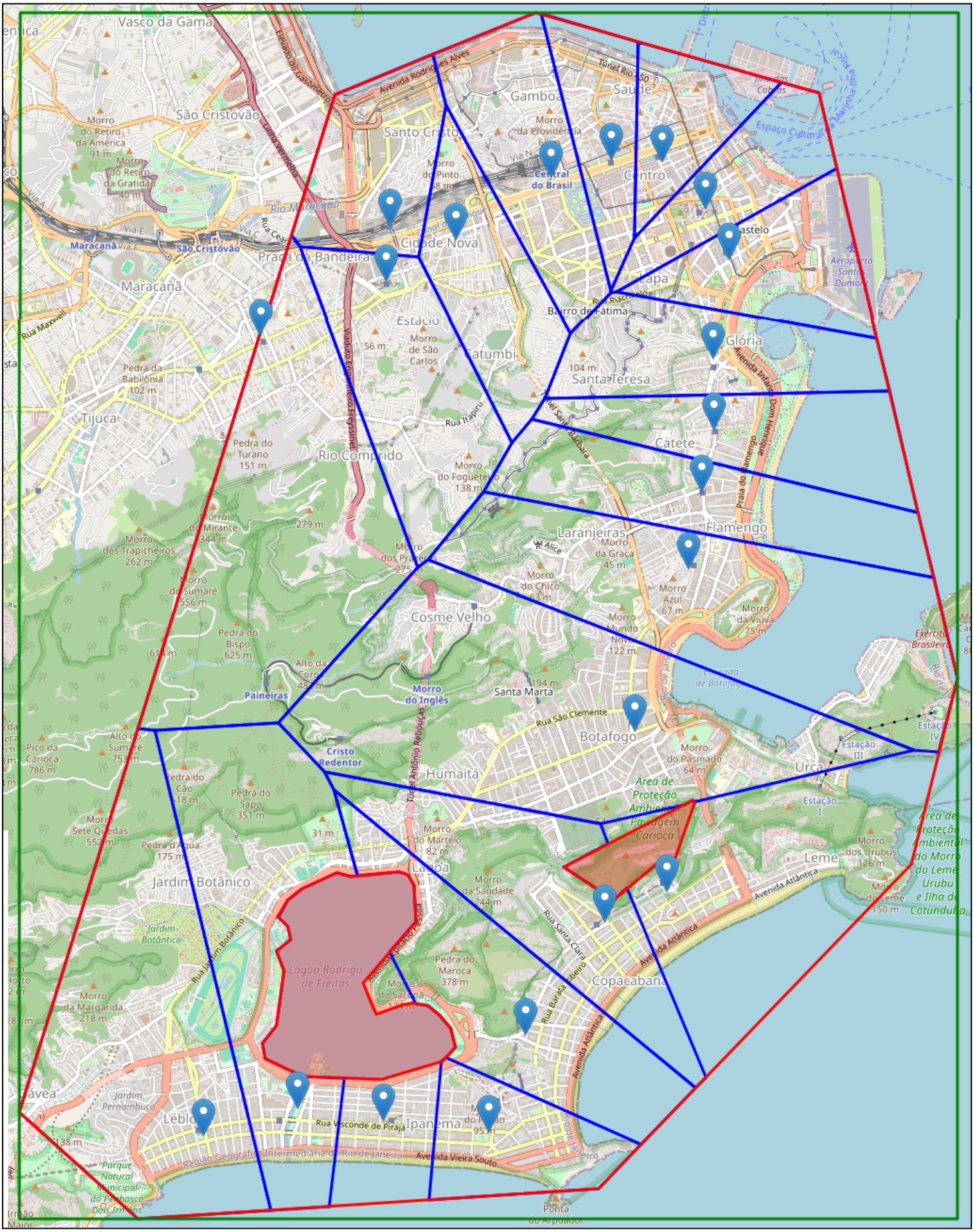
Results of step 2 using the bikes dataset (markers hidden for easy viewing)

3. Clip Voronoi Diagram to outer polygon:
   The Voronoi diagram is clipped to the outer polygon. Since all the seeds are contained in the outer polygon, as a result of step 1, the clipped Voronoi diagram will contain all the seeds.
   However, the outer polygon must be convex, or else a Voronoi cell could break open in a corner of the outer polygon. This is one of the limitations of this approach.

4. Introduce obstacles:
   In this step, we apply the obstacles to the Voronoi diagram. We iterate over the obstacles and for each one we find the cells that intersect with the obstacle and clip these cells to the obstacle. This adapts the cells to contour the obstacles and is equivalent to subtracting the obstacles from the Voronoi diagram.
   I found some limitations in this approach: The obstacles cannot intersect each other or the outer polygon. The inner obstacles should also be convex for the same reason presented in step 3. Non convex polygons do not break the algorithm, but the final Voronoi Diagram may be missing cells.



Final results using bikes dataset (markers hidden for easy viewing)

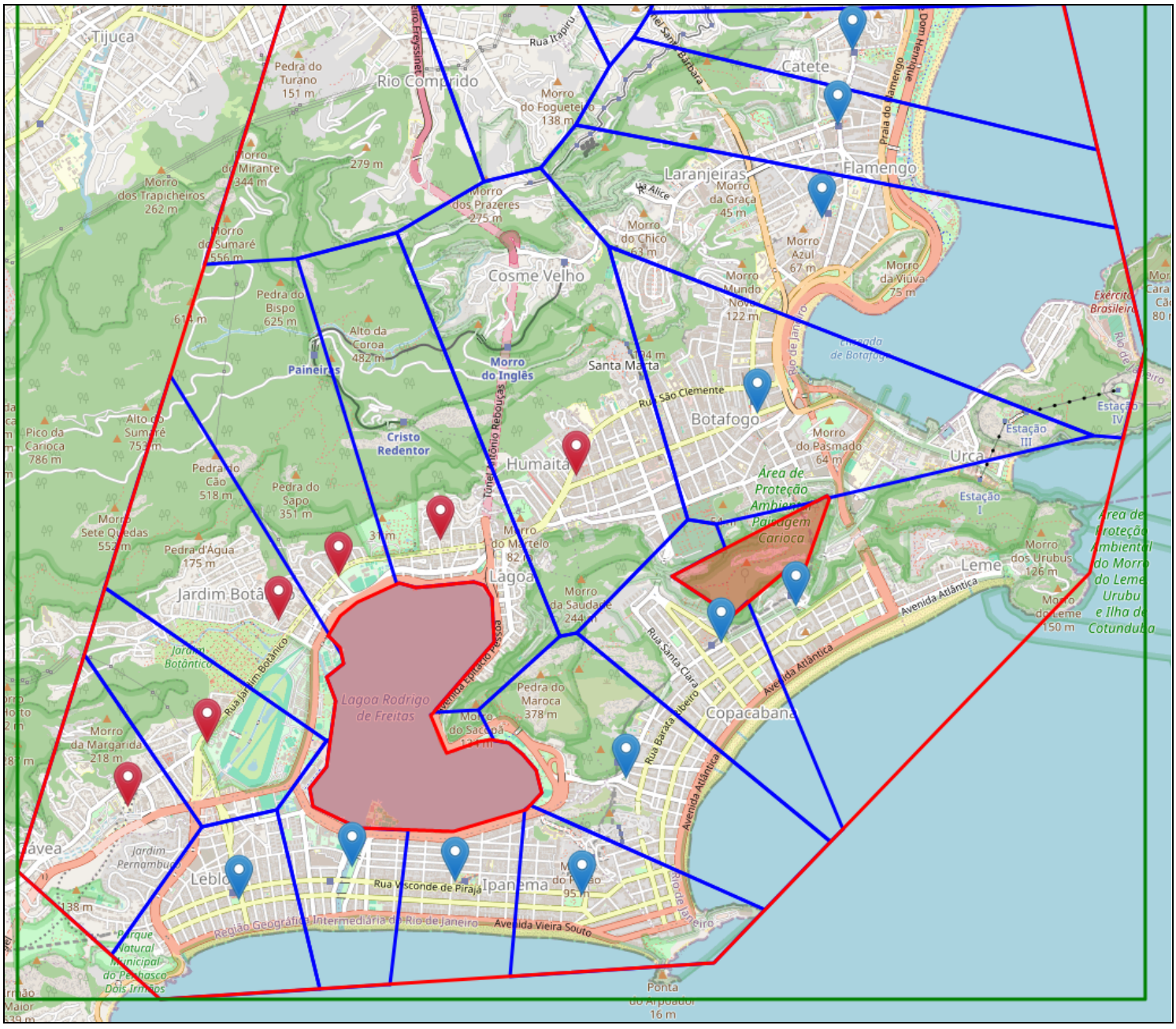


Final results using the metro dataset

Interactive planning of new stations

By overlaying the Voronoi cells onto the map, we gain insights into the spatial distribution and coverage of existing metro and bike-sharing stations. The Voronoi cells act as a geometric representation of influence zones, delineating the areas served by each station. This visualization allows us to identify and quantify gaps in coverage, revealing regions of the city that are not adequately served by the current transportation network. Introducing new stations strategically to these underserved areas becomes a targeted approach for improving urban mobility.

For instance, we could simulate the potential impact of adding a new metro line that traverses large cells, significantly reducing their sizes. This action not only addresses the immediate area but also has a cascading effect, impacting adjacent cells and efficiently optimizing the network's reach. The visual example below illustrates how the addition of a new metro line (depicted in red) intersects and diminishes the size of multiple Voronoi cells, symbolizing the potential for comprehensive improvement in accessibility across the city.



Example: Introduction of a new metro line

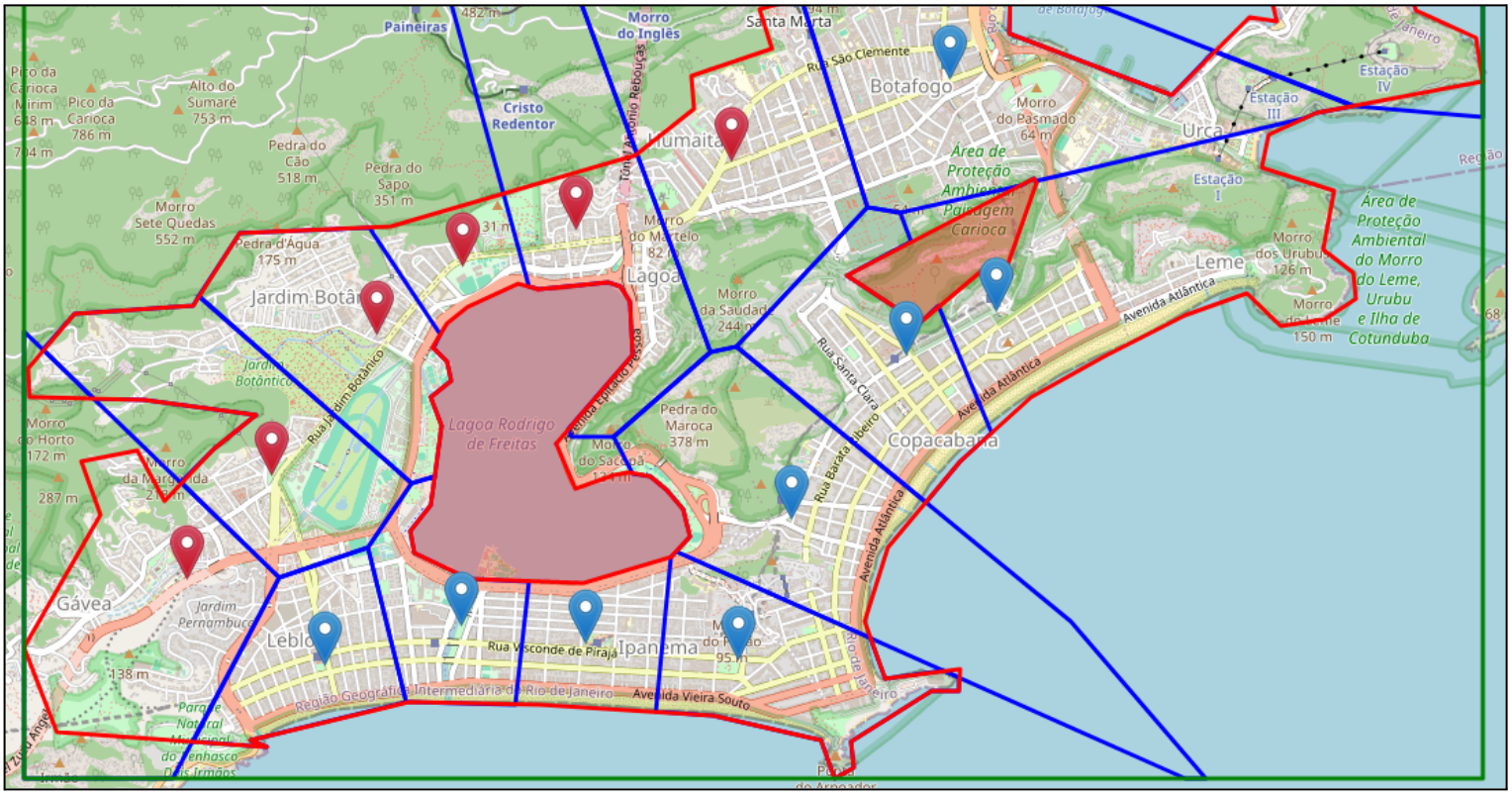
Drawbacks

The presented approach, while straightforward, reveals its limitations when dealing with non-convex polygons. The requirement for convexity in both the outer boundary and inner obstacles poses a significant constraint, limiting the tool's ability to provide a fully accurate representation of urban mobility gaps.

The current model, constrained by these geometric considerations, tends to include sea, mountains, and uninhabited zones within the Voronoi cells.

The image below shows an example of a much more accurate outer boundary. We can see that the algorithm fails to clip the cells that intersect the outer polygon near its non-convex corners (such as the most leftward station, at PUC).

However, when comparing this image with the one from the last page, it is noticeable that the voronoi cells of the second image are much closer to the real populated areas of the city that need to be served by the mobility networks.



An example of invalid voronoi diagram using an non-convex outer boundary

Conclusion

While our current solution reveals notable drawbacks, particularly the reliance on convex polygons for both outer boundaries and inner obstacles, the exploration of alternative methodologies presents promising avenues for improvement. There are two approaches that I would try next.

The first one would be using different distance metrics in the Voronoi calculation, accounting for obstacles, factoring them as infinite barriers, with infinite distances. This way, we would not need to calculate intersections or subtractions of polygons afterwards.

The second approach would be replacing the Voronoi Diagram with a Constrained Delaunay Triangulation, which is a geometric algorithm that generates a triangulated network while respecting pre-defined constraints, ensuring that specific edges or segments are preserved in the resulting triangulation.

In final considerations, my exploration into the intersection of computational geometry and urban mobility planning has unveiled both the potential and challenges inherent in leveraging geometric algorithms for informed decision-making. The development of this project, while exhibiting limitations related to convexity requirements, stands as a testament to the complexities of representing diverse urban landscapes.

References

"Voronoi with Obstacles, Delaunay Constrained Triangulation and Delaunay 3D"
Presentation by Marc Comino Trinidad
https://dccg.upc.edu/people/vera/wp-content/uploads/2014/11/GA2014-DelaunayWithConstrainsDelaunay3D-Marc-Comino.pdf

"Diagrama de Voronoi" Presentation by Waldemar Celes
https://web.tecgraf.puc-rio.br/~celes/docs/inf2604/voronoi.pdf

"Metro Paris Voronoi" application by Vincent Pantaloni
https://www.geogebra.org/m/xM8TRtJK

D3-delaunay Voronoi documentation
https://d3js.org/d3-delaunay/voronoi

Pybikes scraping tool
https://github.com/eskerda/pybikes/tree/master