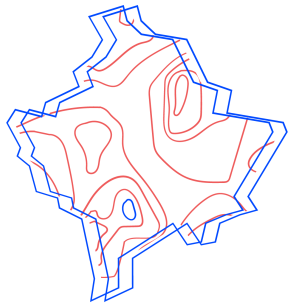
Pedaling Towards Progress

Analyzing Washington, DC's Bikesharing system using Open-source tools

Maxwell Lindsay

Van Oord

June 28, 2023



FOSS4G

Prizren, 2023

About me

- Originally from Washington, DC area
- Background in Coastal Engineering
- Currently work as a Geospatial Developer at Van Oord, In Rotterdam, The Netherlands

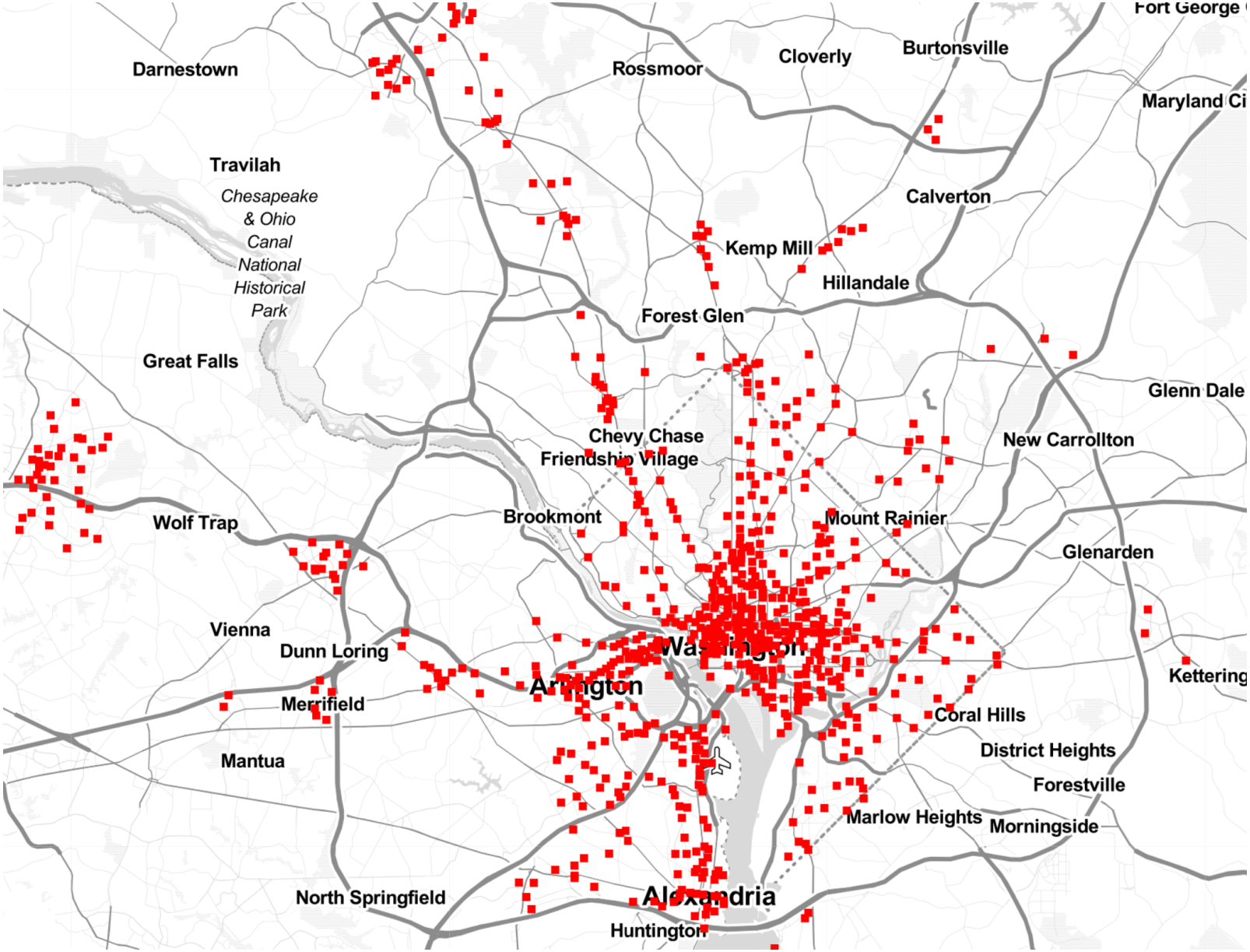
Van Oord

Marine ingenuity

Capital Bikeshare basics

- Docked bikeshare
- Primarily intended for short trips
- 700+ docking stations
- $\approx$ 35 million trips (so far)

The current station map



The data

Capital Bikeshare publishes the following data about each trip on a monthly basis:

- start time
- end time
- start station index
- end station index
- Membership status of the rider (Member or Non-member?)

These data are available as csv files per month or year from Capital Bikeshare at https://ride.capitalbikeshare.com/system-data

Example data

The raw CSV files are available from the Capital Bikeshare Data portal. This is a small example what the data looks like

ride_id	rideable_type	started_at
5F3D280238A782FE	docked_bike	2023-05-12 18:57:
97EC218DACB24849	classic_bike	2023-05-23 07:55
31D19AC7BA317018	electric_bike	2023-05-05 17:27:

Goal 1

Count how many trips ocurred on every unique combination of stations

Goal 2

Find the route of this each of these trips

Goal 3

Combine these two items to get a map of the estimated number of bikeshare trips on every street/road/path in the city

The Challenges

- Efficiently putting hundreds of thousands of trips through a routing engine
- How to combine these hundreds any thousands of complex multiline geometries to sum the trips on each part (without melting my laptop)

Overview of my approach

1. Download all CSV files for all trips
2. Parse and normalize the CSVs using Pandas
3. Find The number of trips between unique pairs of stations
4. Build Valhalla routing tiles
5. Find a route between each station pair using Valhalla
6. Aggregate the trip statistics across every single route

Data cleaning and validation

For each trip, the trip time is calculated. Trips are considered invalid based on the following criteria:

- Longer than a 4 hours
- Starting and ending at the same station
- Stations with an invalid start or end station

As of May 2023 there are 35,231,413 trips. 175,603 are longer than 4 hours and are removed.

Find unique trips between station pairs

$$35,136,810 \text{ trips} \rightarrow 183,959 \text{ station combinations}$$

If we drop trips that start and end at the same dock, we are down to 183,214 trips. If we consider only *which two* stations are invovled:

$$183,214 \rightarrow 105,636$$

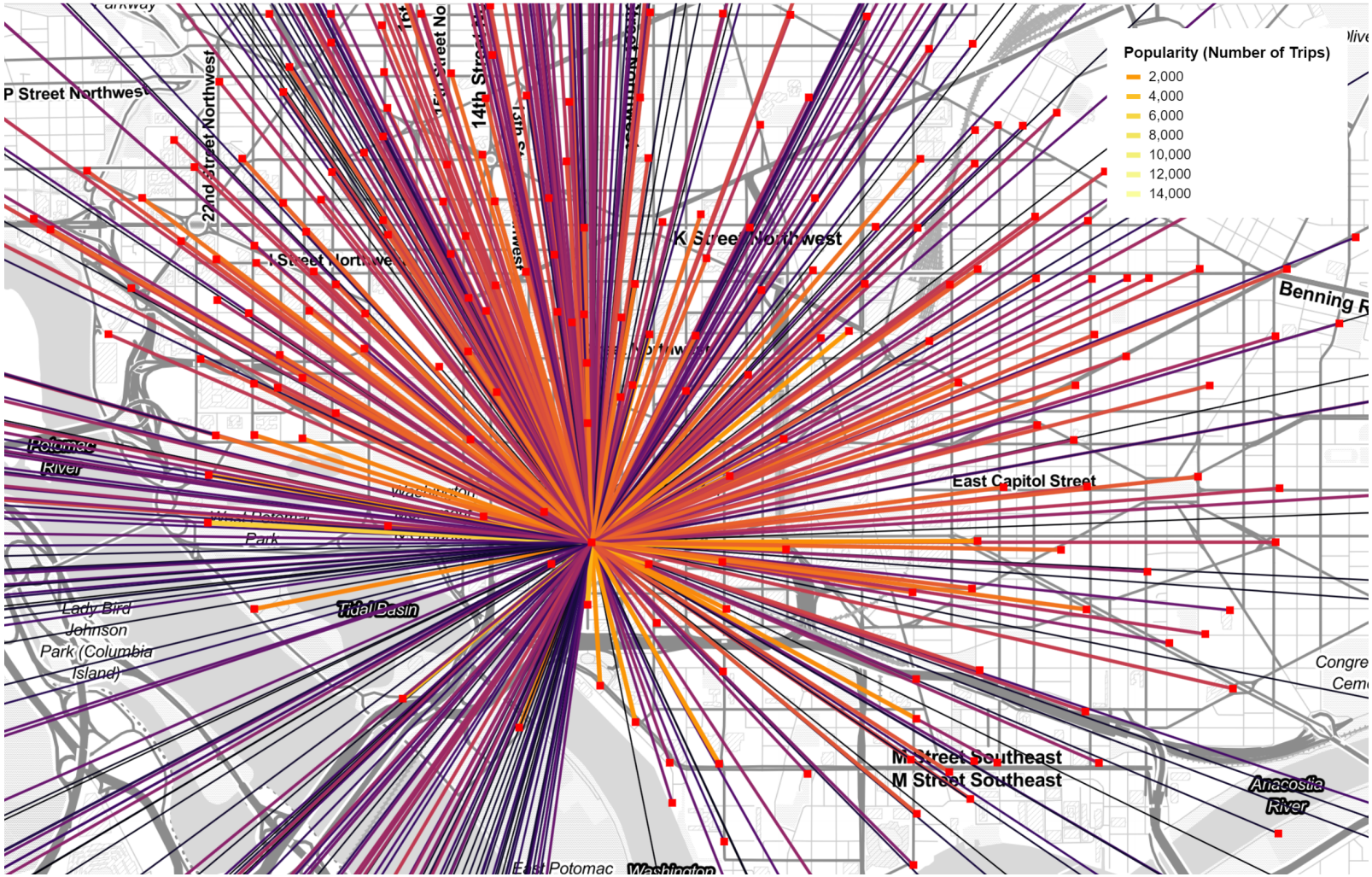
These trips are the final ones to route

Find the number of trips between any 2 stations

Implemented by looping over the individual trips in a Pandas DataFrame.

Maxwell Lindsay

Results of trips between stations

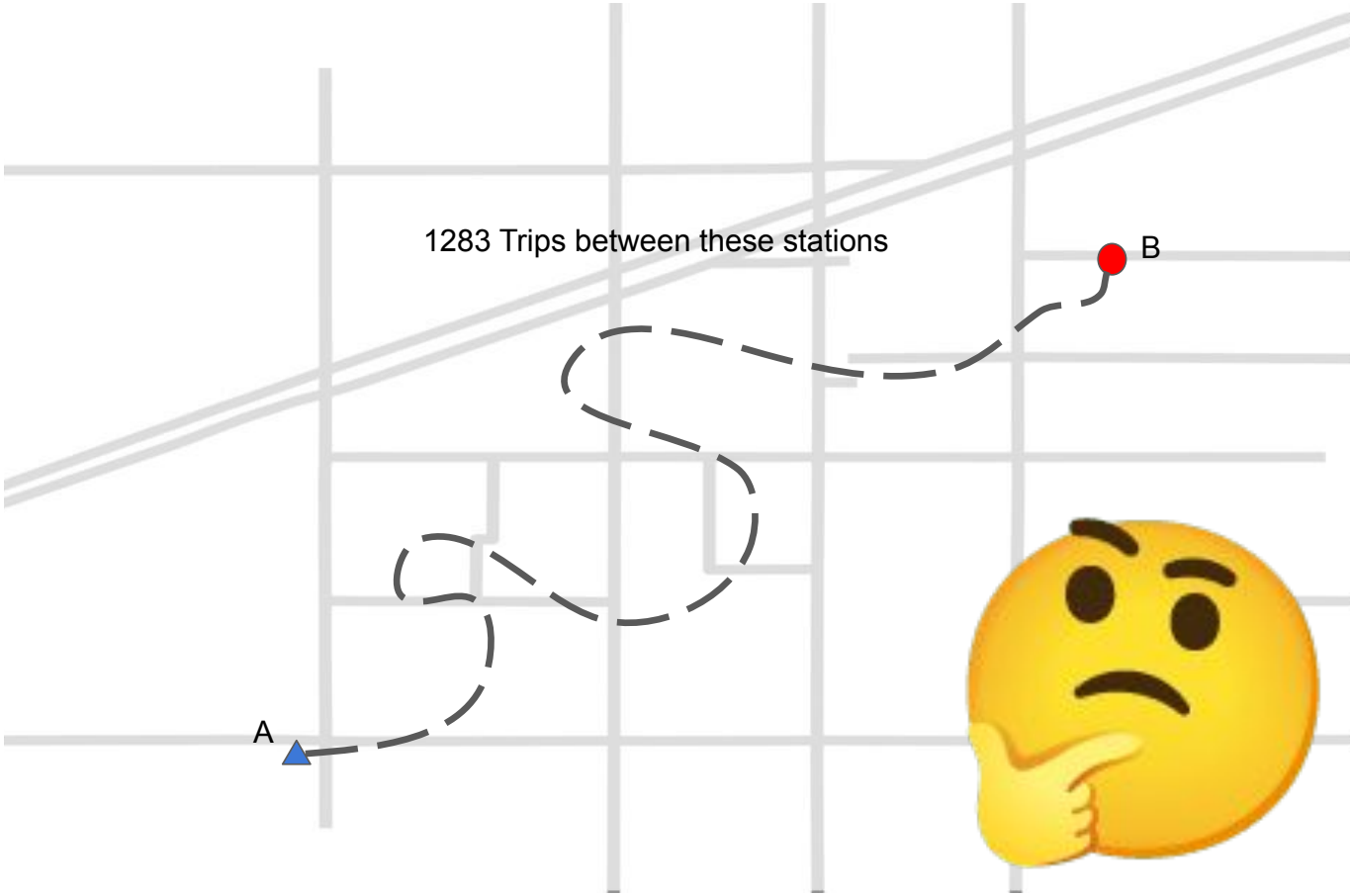


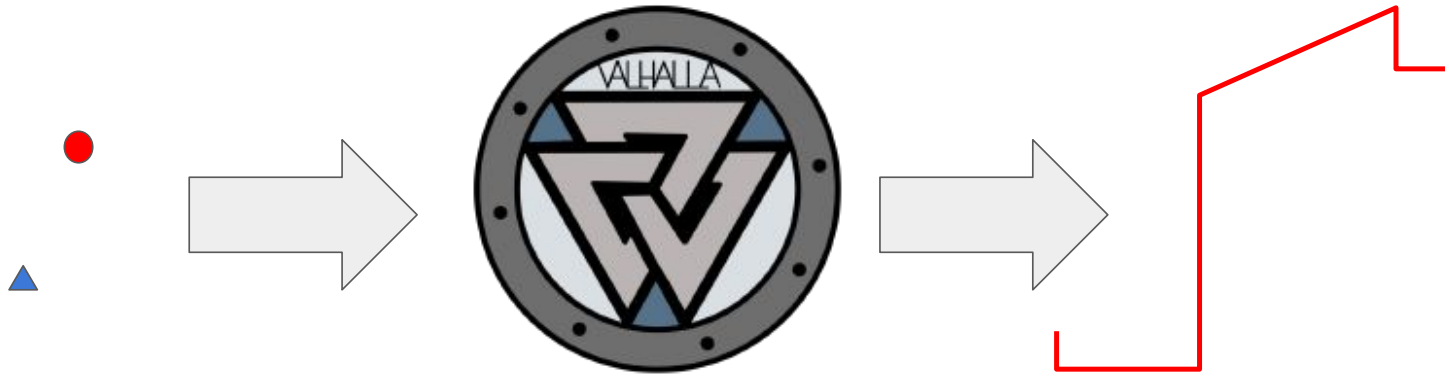
Building a routing network from OSM data

- Download the OSM data export from geoFabrik in protobuffer format
- Trim and merge the protobuf OSM files
- Let Valhalla Docker container build the routing tiles automatically

Routing these trips

Back to Goal 2





Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type

Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type
- Willingness to share roads with cars

Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type
- Willingness to share roads with cars
- Willingness to bike up hills

Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type
- Willingness to share roads with cars
- Willingness to bike up hills
- Average cycling speed

Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type
- Willingness to share roads with cars
- Willingness to bike up hills
- Average cycling speed

Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type
- Willingness to share roads with cars
- Willingness to bike up hills
- Average cycling speed

How to do this efficiently?

Bike Routing Considerations

Vallhalla allows very granular control over bicycle routing, allowing us to set parameters for a cyclist's:

- Bicycle Type
- Willingness to share roads with cars
- Willingness to bike up hills
- Average cycling speed

How to do this efficiently?
Avoid the Valhalla HTTP API and use *pyvalhalla*

Back to Goal 3

How to combine it all

The challenge:

Back to Goal 3

How to combine it all

The challenge:
How to join over 100k linestrings, summing a certain value *only where they overlap*

Back to Goal 3

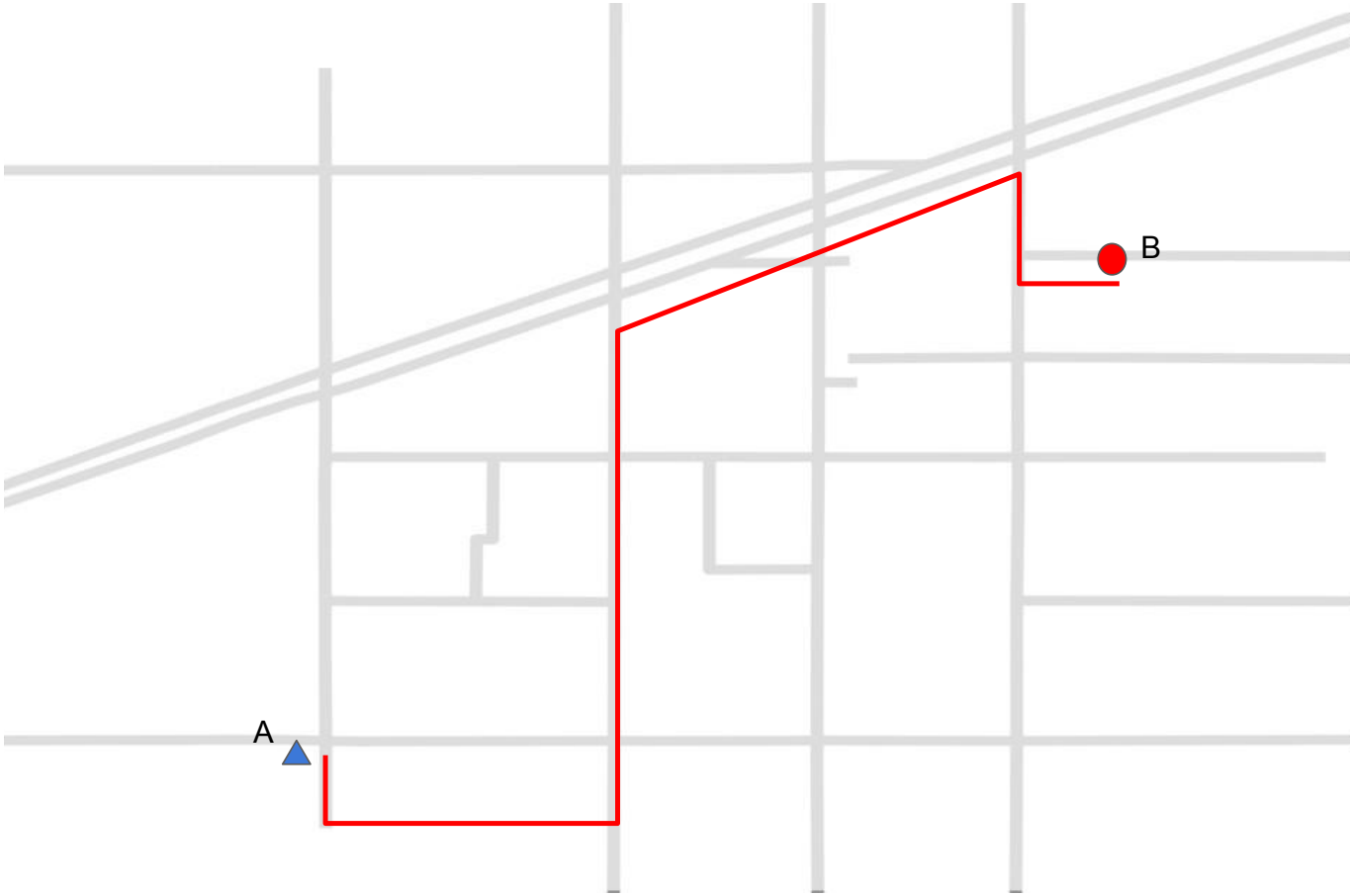
How to combine it all

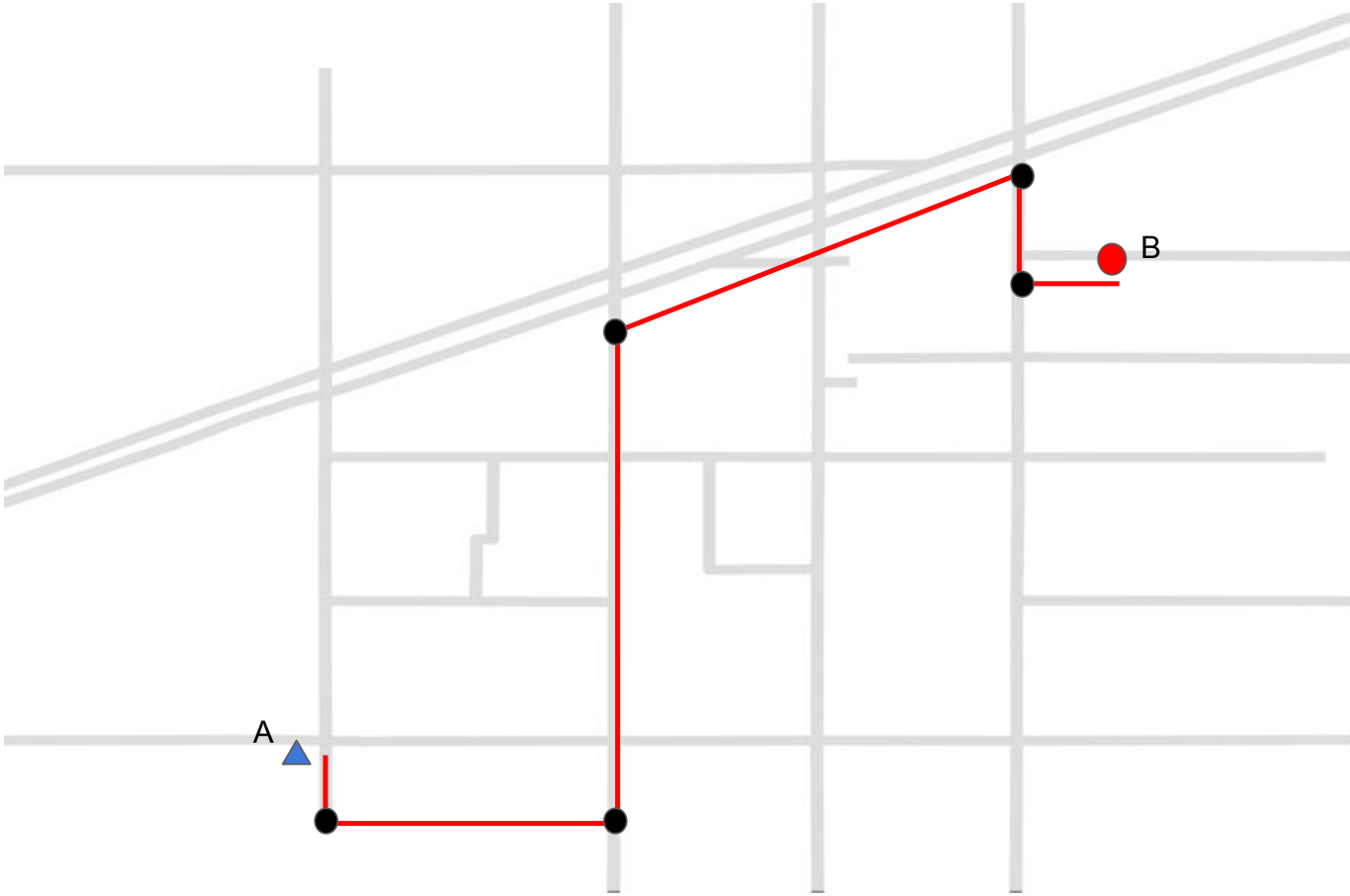
The challenge:
How to join over 100k linestrings, summing a certain value *only where they overlap*
The solution I found:

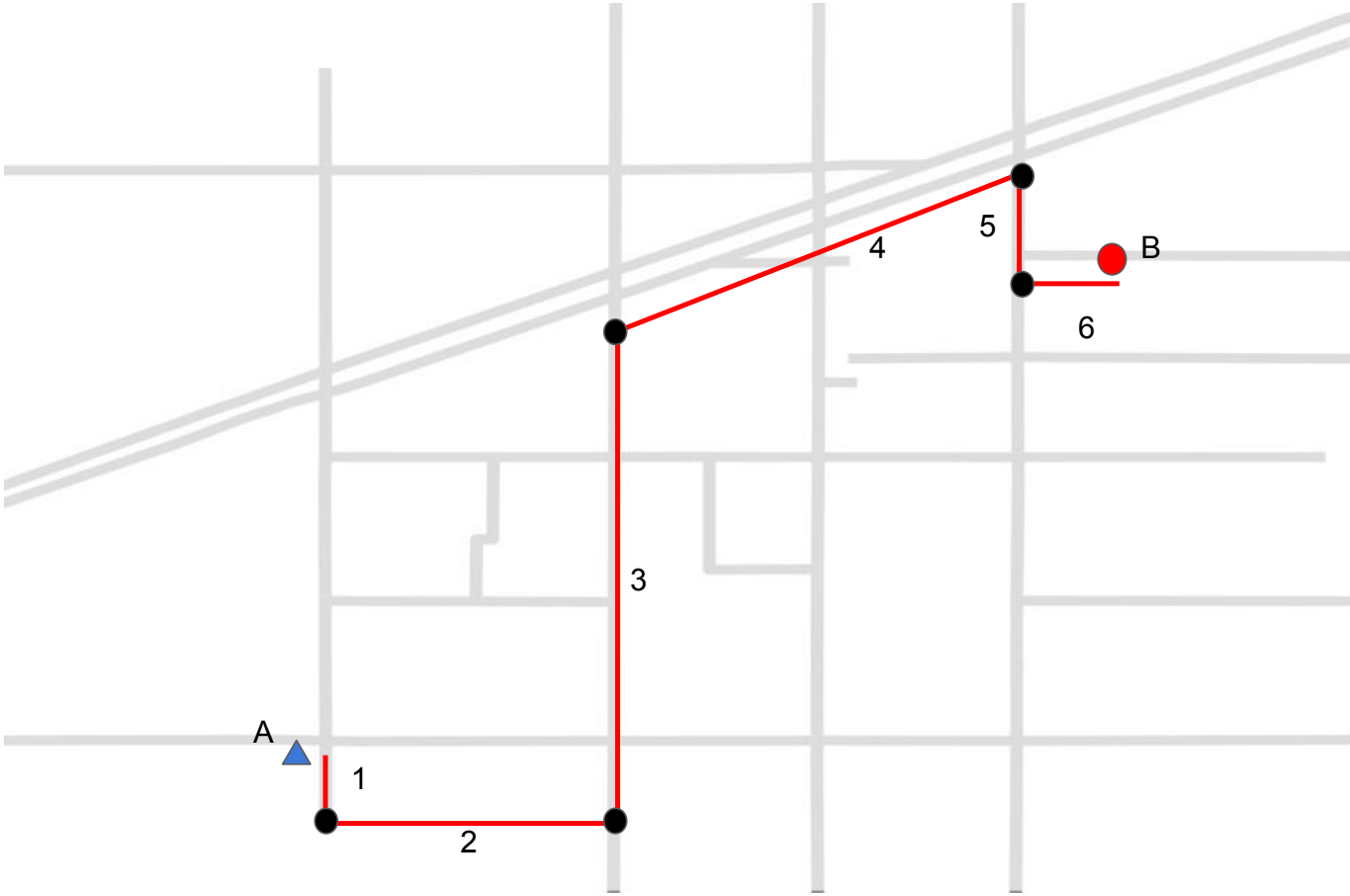
Back to Goal 3

How to combine it all

The challenge:
How to join over 100k linestrings, summing a certain value *only where they overlap*
The solution I found:
TopoGeometry

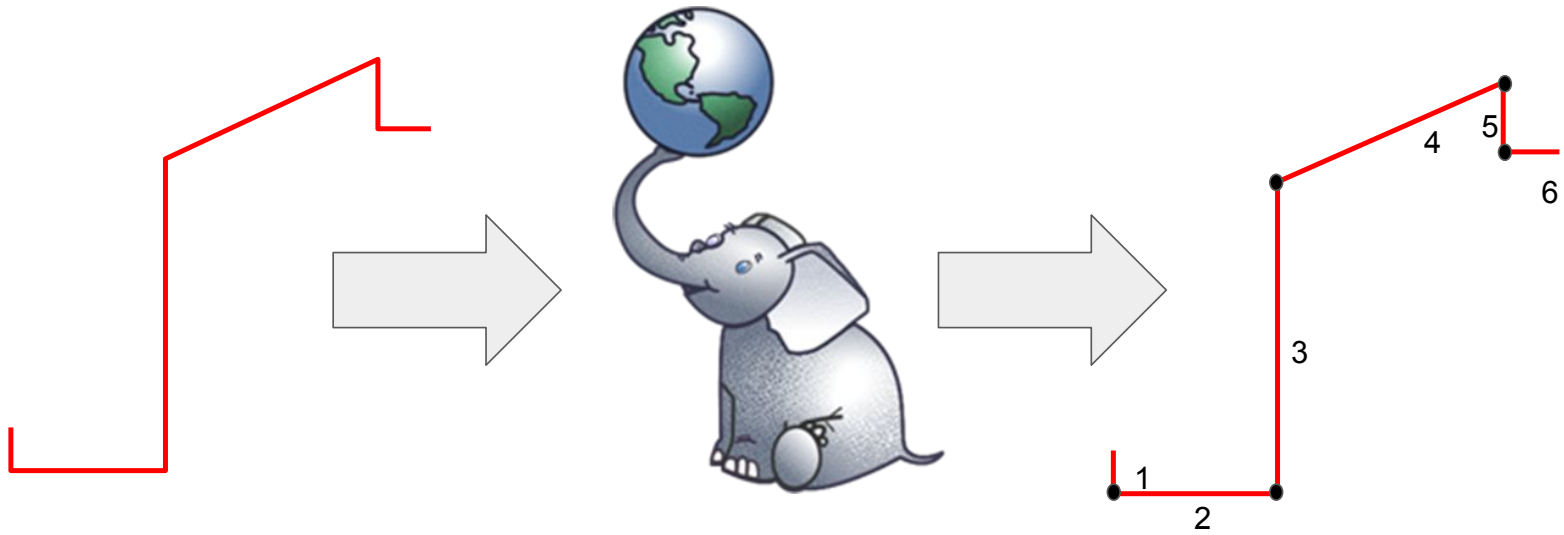

B
A


B
A


A
1
2
3
4
5
6
B

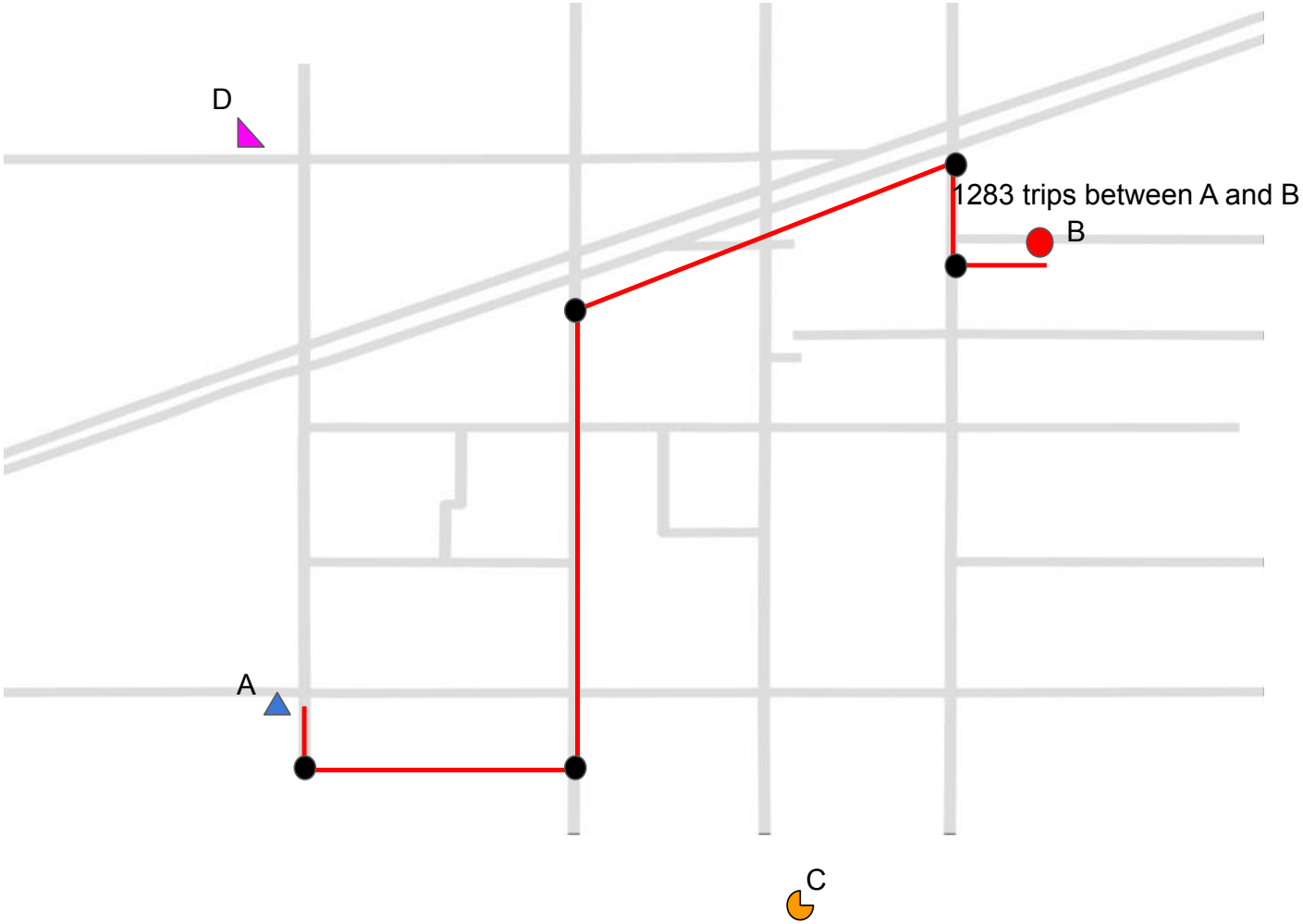
PostGIS topology

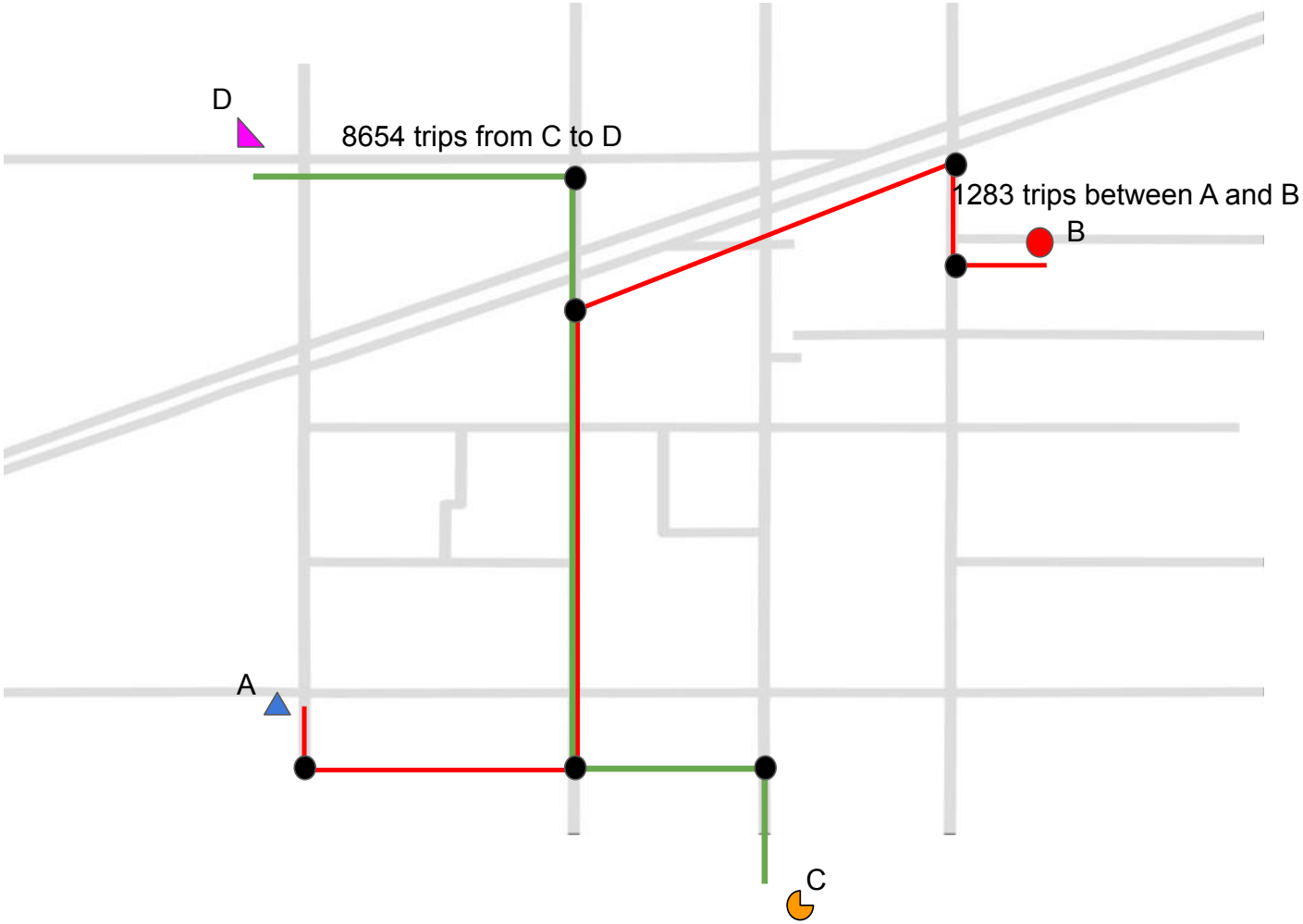
- Using PLpgSQL triggers, add a corresponding entry to the topogeometry table for every new entry to the routes tables

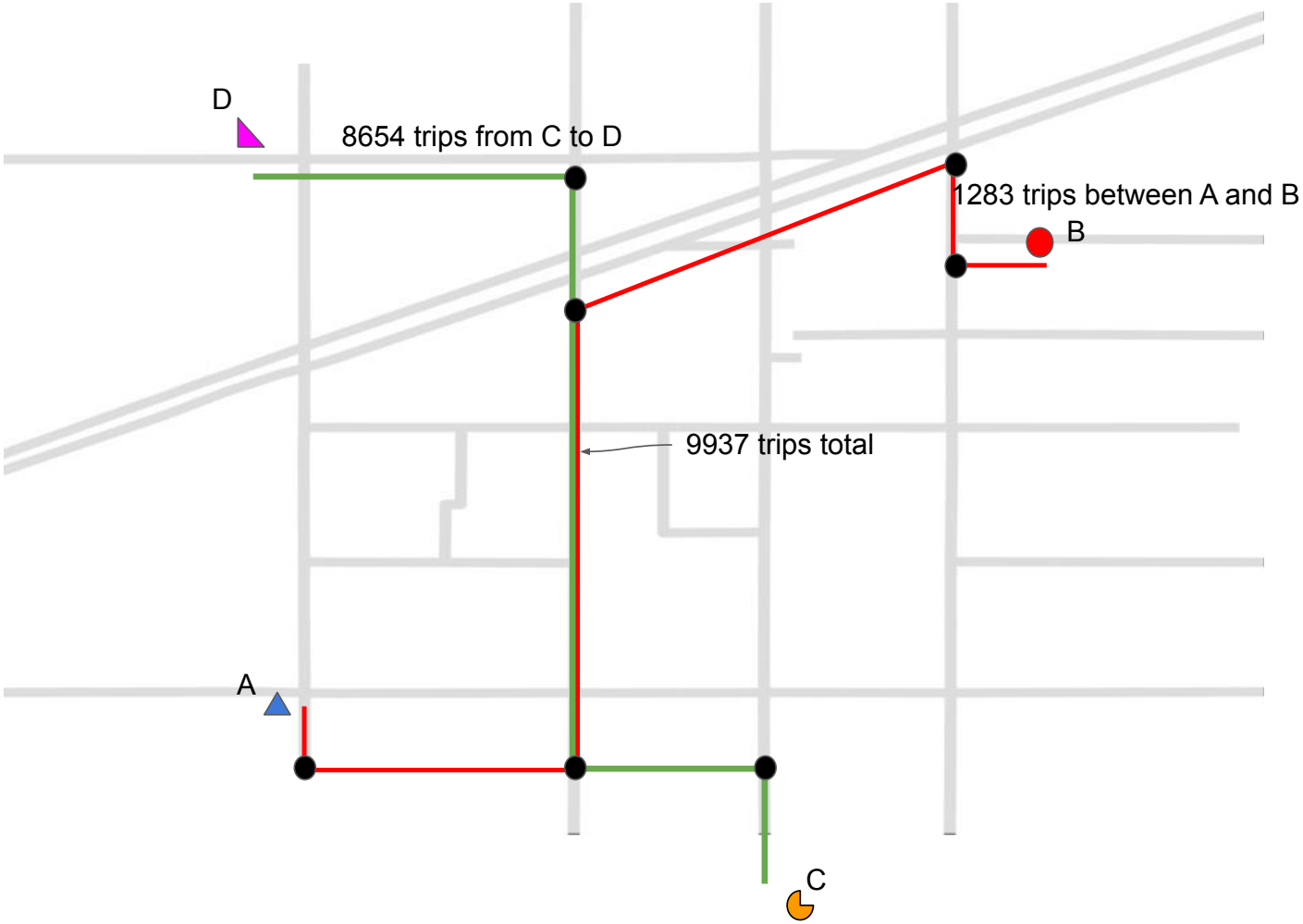

1
2
3
4
5
6

Sum up the trips on every topological edge

- Using an SQL query, we can sum the trips on every unique ”section” of road in DC




D
8654 trips from C to D
1283 trips between A and B
B
A
C


D
8654 trips from C to D
1283 trips between A and B
B
9937 trips total
A
C

The actual implementation

- PostGIS hosted inside a Docker container

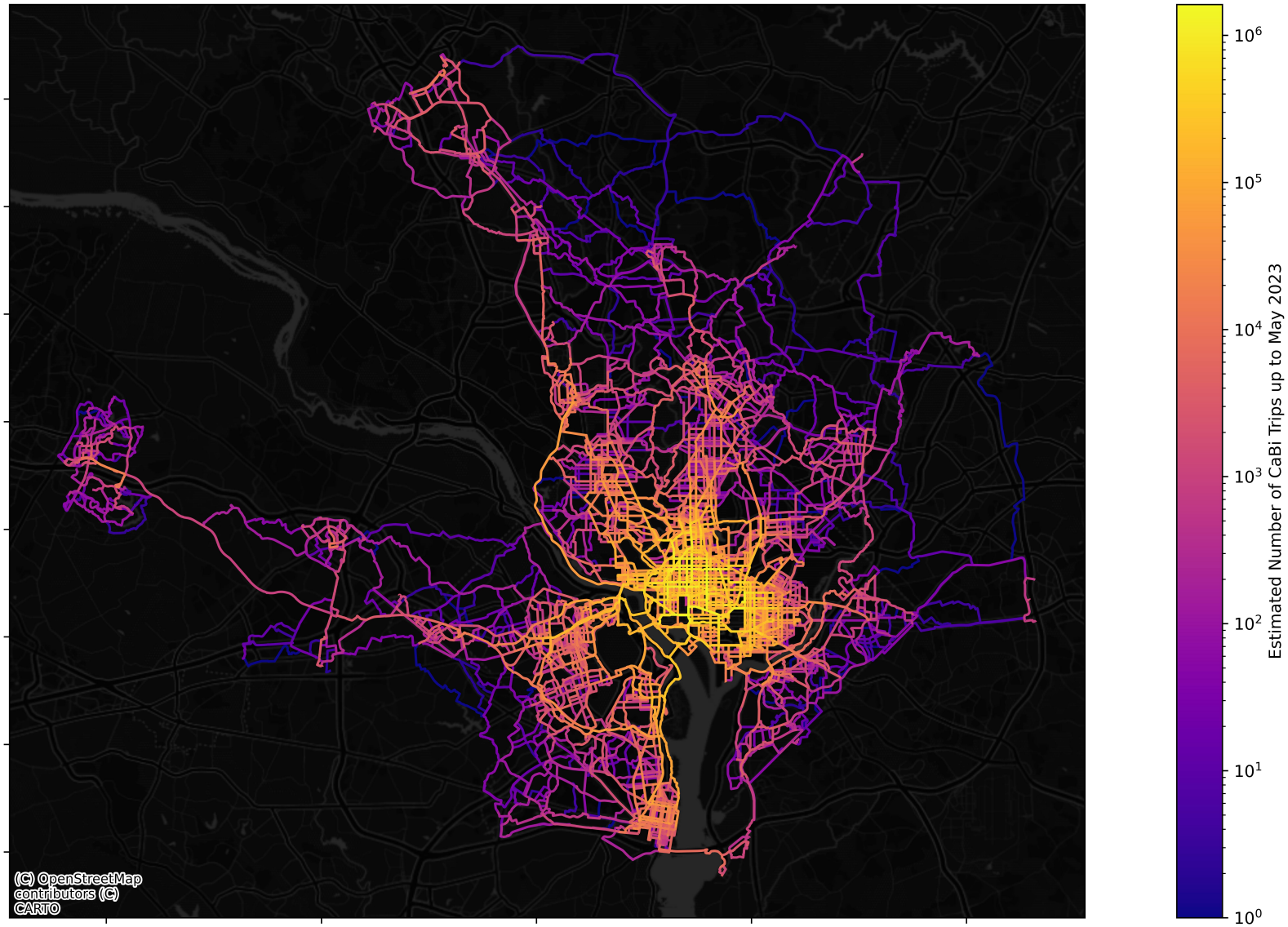
The actual implementation

- PostGIS hosted inside a Docker container
- A series of Python scripts load the data, and run the routing software

The actual implementation

- PostGIS hosted inside a Docker container
- A series of Python scripts load the data, and run the routing software
- Makefile ties everything together

Results


(C) OpenStreetMap contributors (C) CARTO
Estimated Number of CaBi Trips up to May 2023
10^6
10^5
10^4
10^3
10^2
10^1
10^0

Validation

How accurate is this?

Are the sums for any individual routes anywhere near accurate?

Validation

How accurate is this?

Are the sums for any individual routes anywhere near accurate?
Probably not?

Which steps made this practical to run on a laptop?

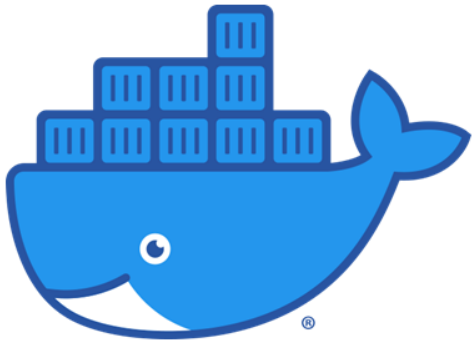
- using *pyvalhalla* instead of hosting it in Docker
- Using topology to sum trips by street
- Using SQL triggers for creating the topogeometry

What would I have done differently?

- More in SQL, less in Python
- Python is perfect for being the ”Glue” and interacting with the various parts

Shoutouts

GeoPandas



Thanks for your interest