Towards a Distributed Mobility Database System

Contact: Esteban Zimanyi (ezimanyi@ulb.ac.be)
Mahmoud SAKR (mahmoud.sakr@ulb.ac.be)

MobilityDB: Architecture

MobilityDB

PostGIS

PostgreSQL

tgeompoint, tgeogpoint,
tint, tfloat, ttext, tbool

geometry, geography

numeric, monetary, character,
data/time, boolean, enum,
arrays, range,
XML, JSON, ...

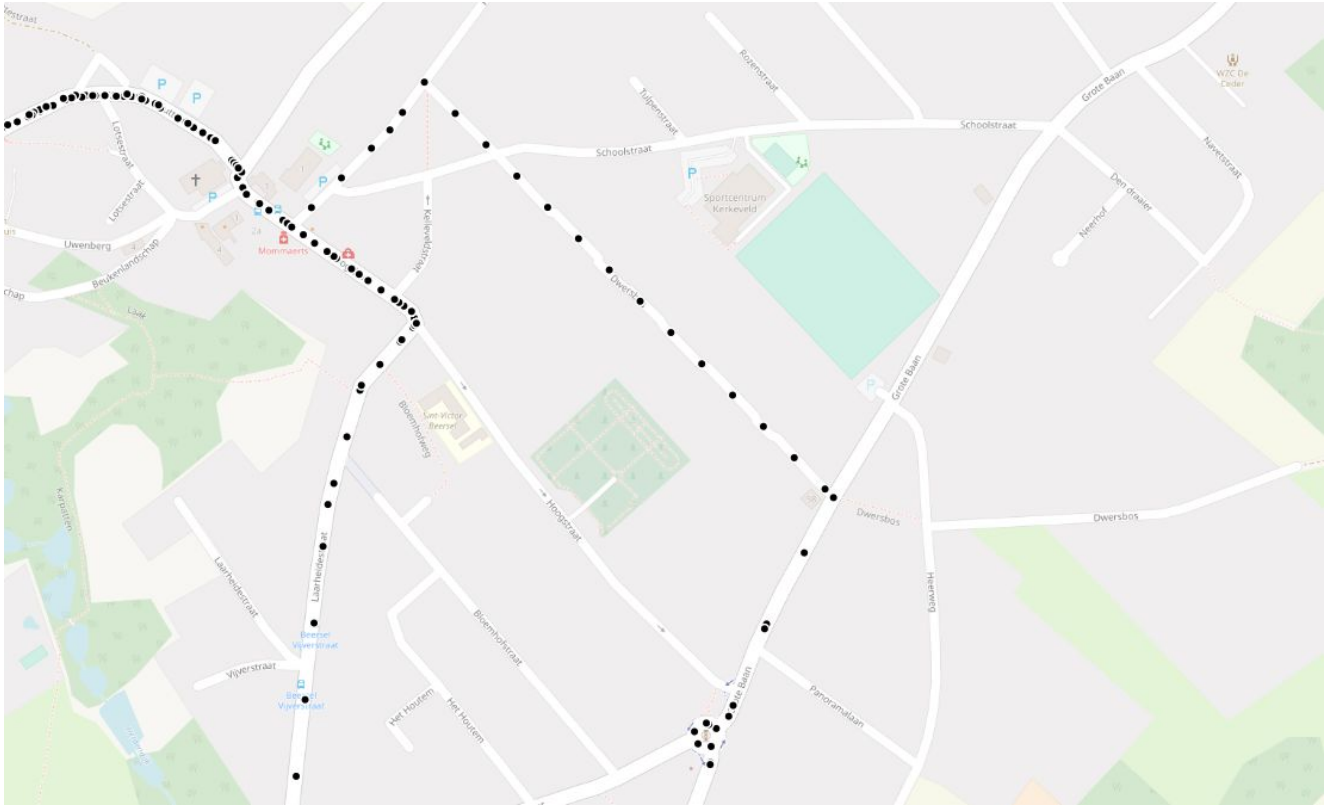
MobilityDB Benefits

- Compact data storage
- Rich mobility analytics
- Big data scale and performance
- Easy-to-use full SQL interface
- Compatible with the huge PostgreSQL ecosystem
- Open source

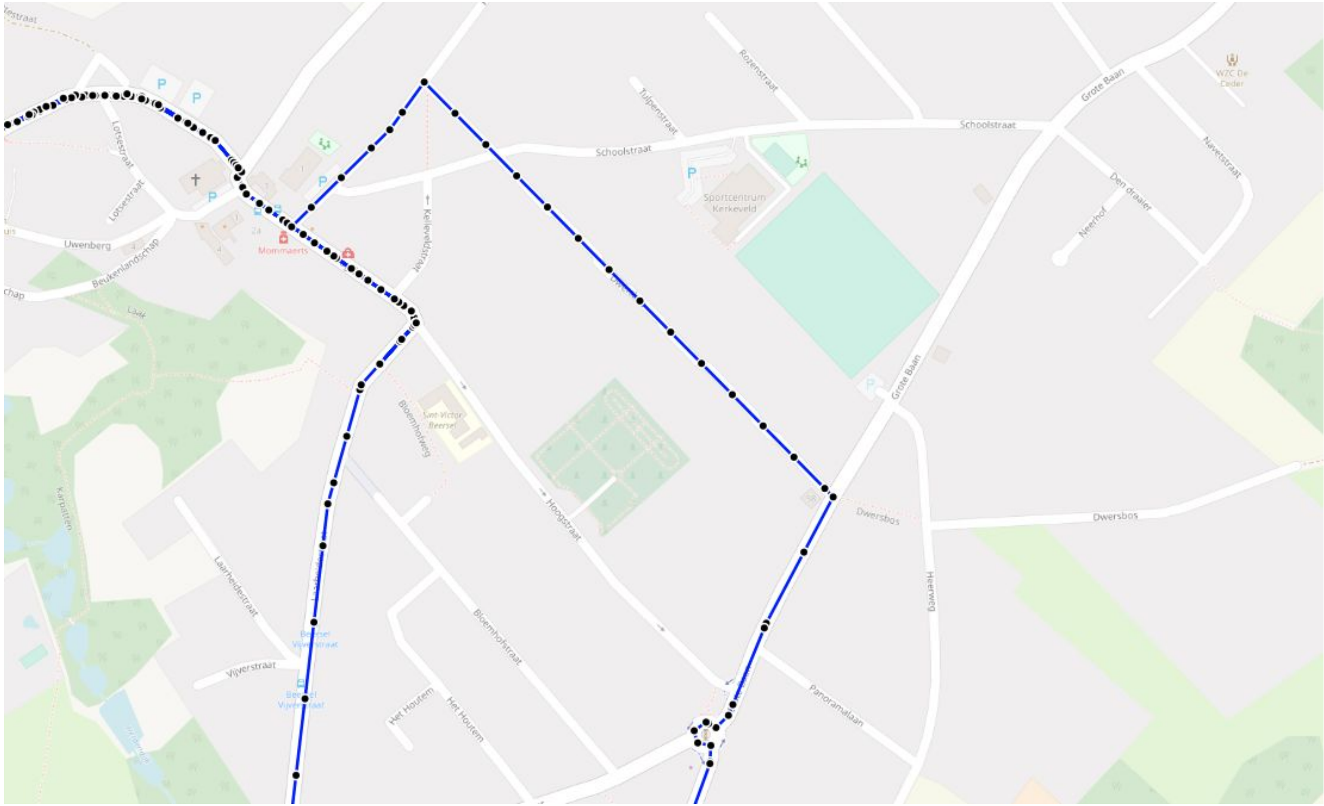
Mobility Data: GPS, DCM

	vehicle integer	day date	seq integer	source bigint	target bigint	t timestamp with time zone	geom text
370	1	2020-06-01	1	16690	34728	2020-06-01 10:01:20.978+02	POINT(490985.514477288 6604283.85305868)
371	1	2020-06-01	1	16690	34728	2020-06-01 10:01:28.081029+02	POINT(490956.213031904 6604299.02553383)
372	1	2020-06-01	1	16690	34728	2020-06-01 10:01:30.978+02	POINT(490940.739620434 6604303.45266601)
373	1	2020-06-01	1	16690	34728	2020-06-01 10:01:34.491879+02	POINT(490921.971156538 6604308.82255297)
374	1	2020-06-01	1	16690	34728	2020-06-01 10:01:39.062744+02	POINT(490911.18429788 6604299.23736115)
375	1	2020-06-01	1	16690	34728	2020-06-01 10:01:40.978+02	POINT(490901.048941147 6604301.05718891)
376	1	2020-06-01	1	16690	34728	2020-06-01 10:01:42.592551+02	POINT(490892.504887324 6604302.5912944)
377	1	2020-06-01	1	16690	34728	2020-06-01 10:01:47.131132+02	POINT(490872.022101019 6604317.29566003)
378	1	2020-06-01	2	34728	16690	2020-06-01 17:53:26.791+02	POINT(490872.022101019 6604317.29566003)
379	1	2020-06-01	2	34728	16690	2020-06-01 17:53:31.929581+02	POINT(490892.504887324 6604302.5912944)
380	1	2020-06-01	2	34728	16690	2020-06-01 17:53:36.791+02	POINT(490910.008155174 6604299.44854041)
381	1	2020-06-01	2	34728	16690	2020-06-01 17:53:37.117666+02	POINT(490911.18429788 6604299.23736115)
382	1	2020-06-01	2	34728	16690	2020-06-01 17:53:39.828856+02	POINT(490921.971156538 6604308.82255297)
383	1	2020-06-01	2	34728	16690	2020-06-01 17:53:46.239706+02	POINT(490956.213031904 6604299.02553383)
384	1	2020-06-01	2	34728	16690	2020-06-01 17:53:46.791+02	POINT(490958.875710594 6604297.64678183)
385	1	2020-06-01	2	34728	16690	2020-06-01 17:53:56.791+02	POINT(491007.174416728 6604272.63740401)
386	1	2020-06-01	2	34728	16690	2020-06-01 17:54:06.791+02	POINT(491055.473122862 6604247.62802619)
387	1	2020-06-01	2	34728	16690	2020-06-01 17:54:10.336527+02	POINT(491072.597559528 6604238.76088375)

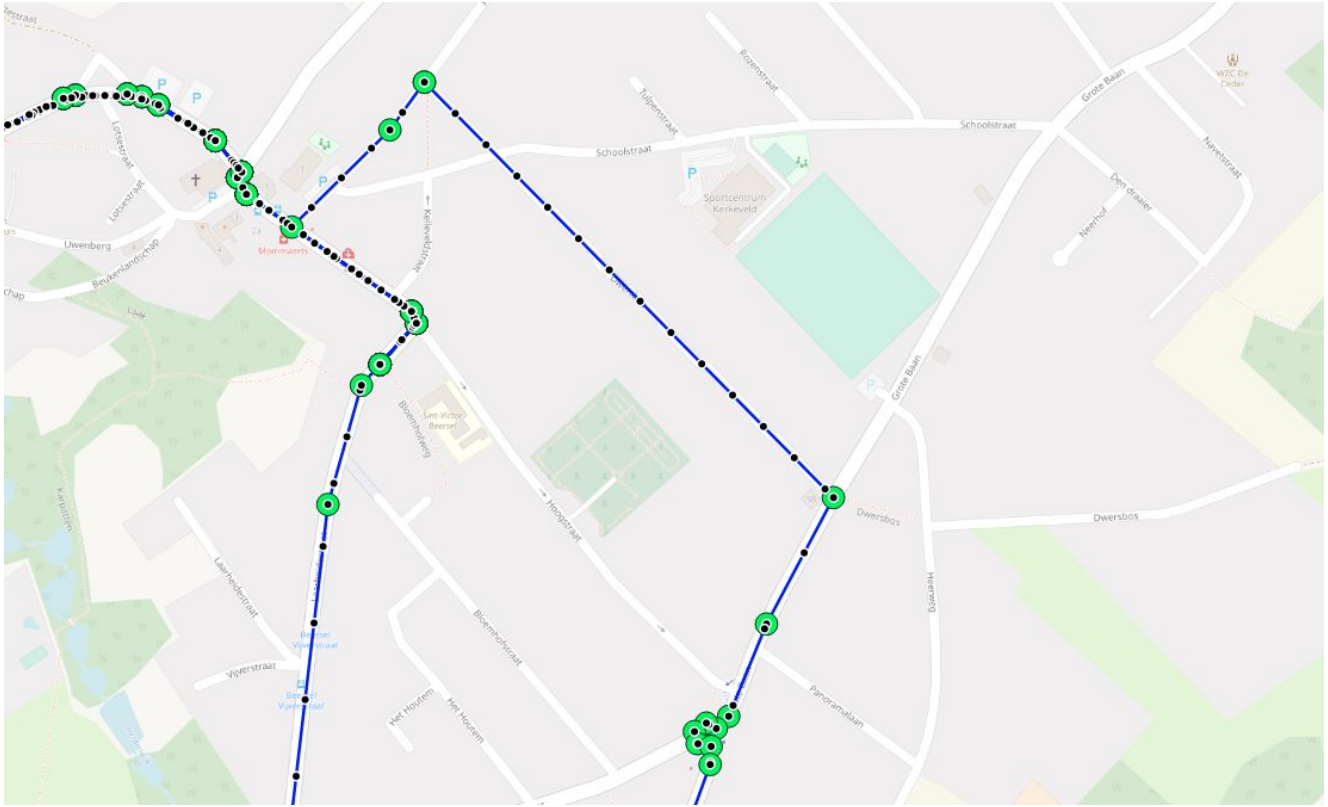
Mobility Data: GPS, DCM



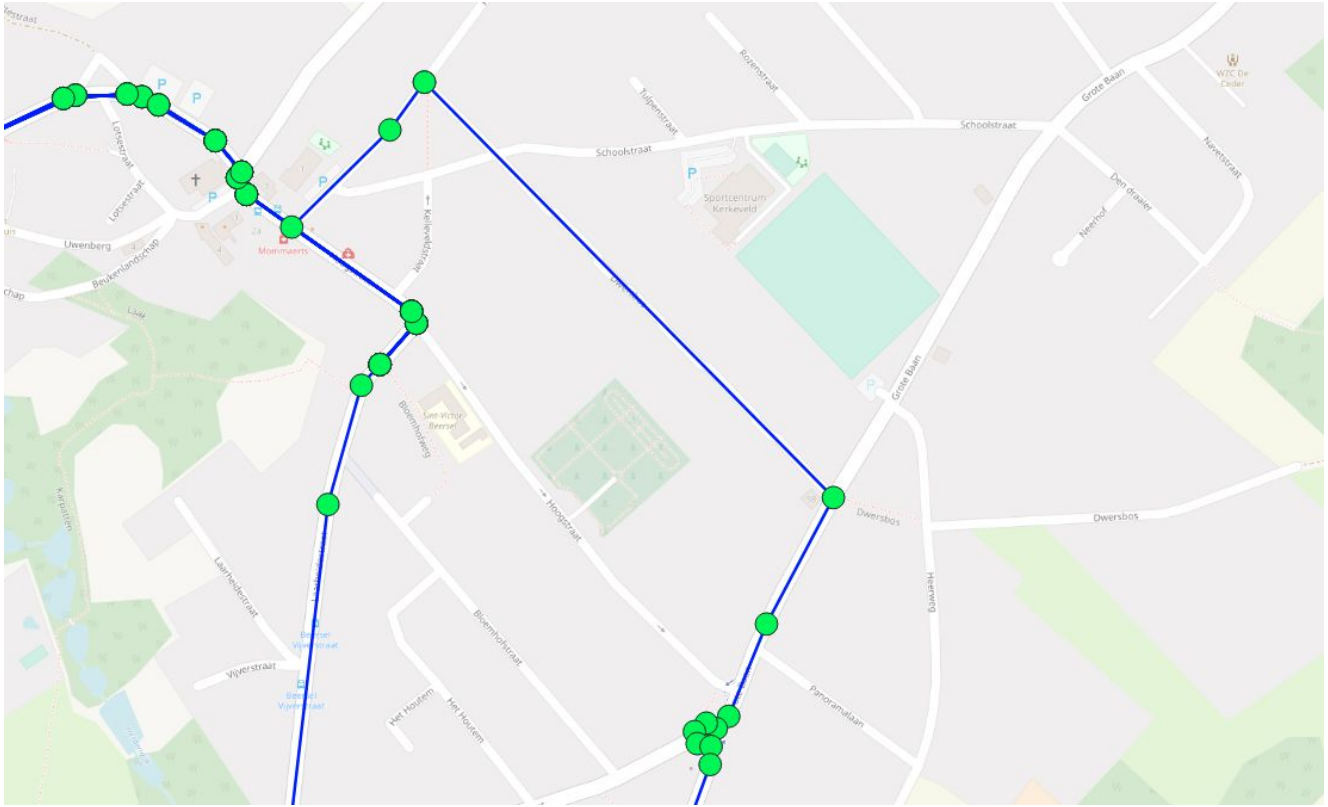
MobilityDB Compact Format



MobilityDB Compact Format



MobilityDB Compact Format



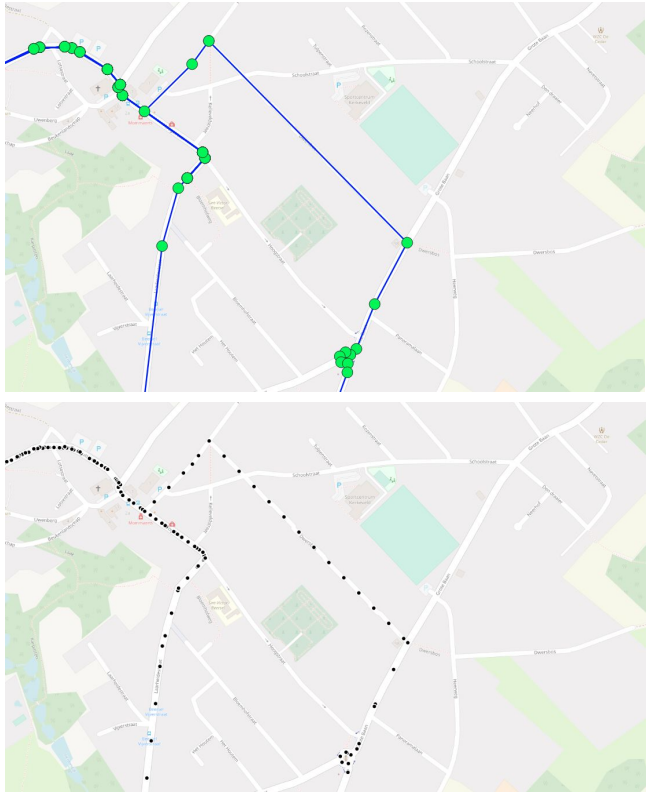
MobilityDB Compact Format

Improvement over existing tools

- 10 - 100x reduction in size
- From millions of rows into thousands
- Query speedup

Novel benefits

- Interpolation
- Native database types
- 300+ implemented algorithms
- Significant reduction in development time



PostGIS -> MobilityDB migration

44384	2015-04-06 06:38:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:38:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:39:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:39:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:40:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:40:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:41:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:41:30	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:42:00	POINT(37.3826816 55.7937783)
44384	2015-04-06 06:42:30	POINT(37.3826816 55.7937783)

44384	[POINT(37.3826816 55.7937783)@2015-04-06 06:38:00+03, POINT(37.3826816 55.7937783)@2015-04-06 0...
44399	[POINT(37.6126166 55.7274032)@2015-04-06 07:14:29+03, POINT(37.6118683 55.7274732)@2015-04-06 0...
44399	[POINT(37.6127783 55.7265099)@2015-04-06 05:32:14+03, POINT(37.6127783 55.7265099)@2015-04-06 0...
62736	[POINT(37.6078283 55.7158566)@2015-04-06 05:35:17+03, POINT(37.607475 55.71504)@2015-04-06 05:35:...
62771	[POINT(37.6124233 55.7264416)@2015-04-06 05:07:57+03, POINT(37.6124233 55.7264416)@2015-04-06 0...
67756	[POINT(37.608135 55.7163933)@2015-04-06 04:47:23+03, POINT(37.60777 55.7153983)@2015-04-06 04:47:...
67762	[POINT(37.6093483 55.7190449)@2015-04-06 16:58:07+03, POINT(37.6094966 55.7188982)@2015-04-06 1...
67762	[POINT(37.6099266 55.7209099)@2015-04-06 04:41:30+03, POINT(37.60921 55.7190516)@2015-04-06 04:4...

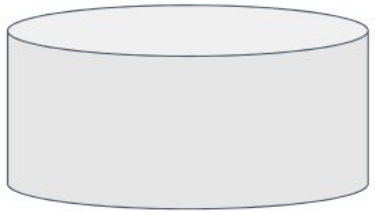
10 billion rows a day

> 500 MB per day

~ 300 GB per year

15 thousand rows

~ 5 MB per day



2GB per year

Temporal Aggregations: Travel Time

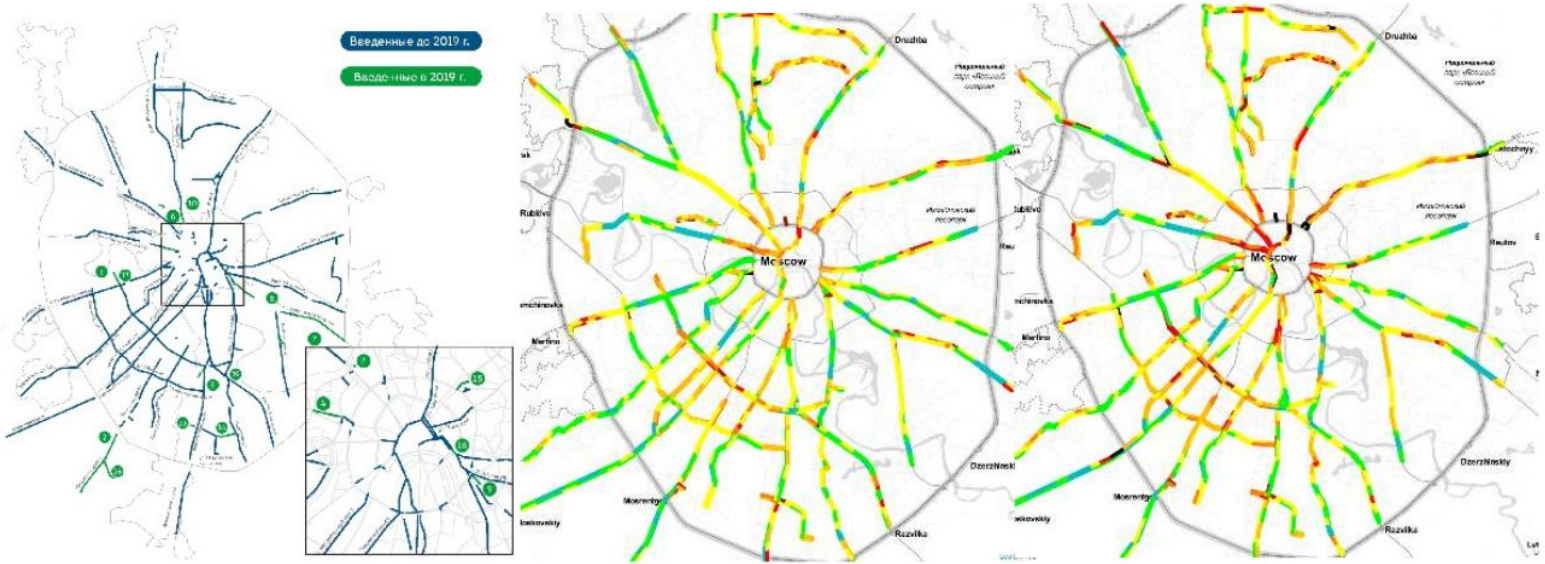
Trolleybus №49

September 2017

September 2018

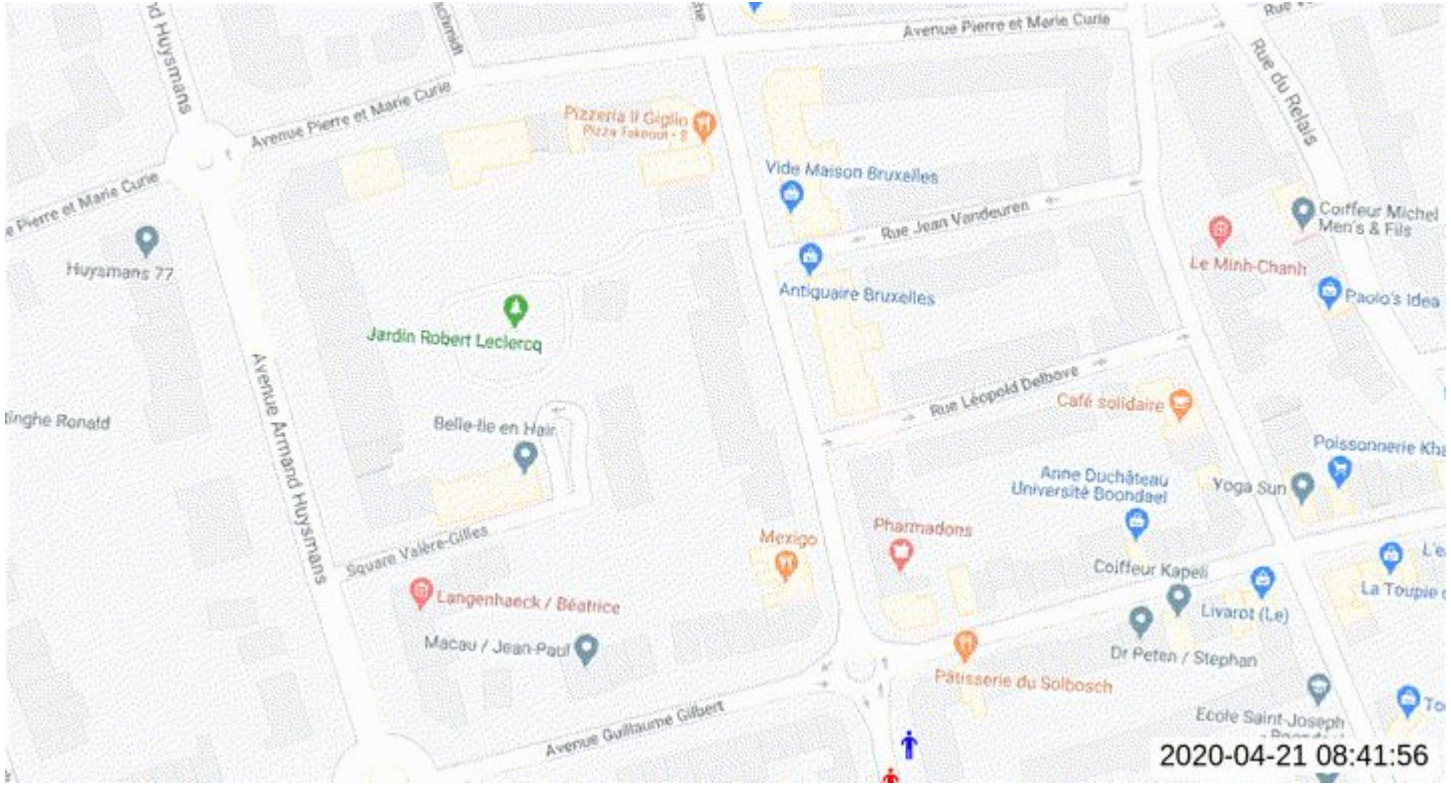
Temporal Aggregations: Velocity Maps

Moscow bus lanes



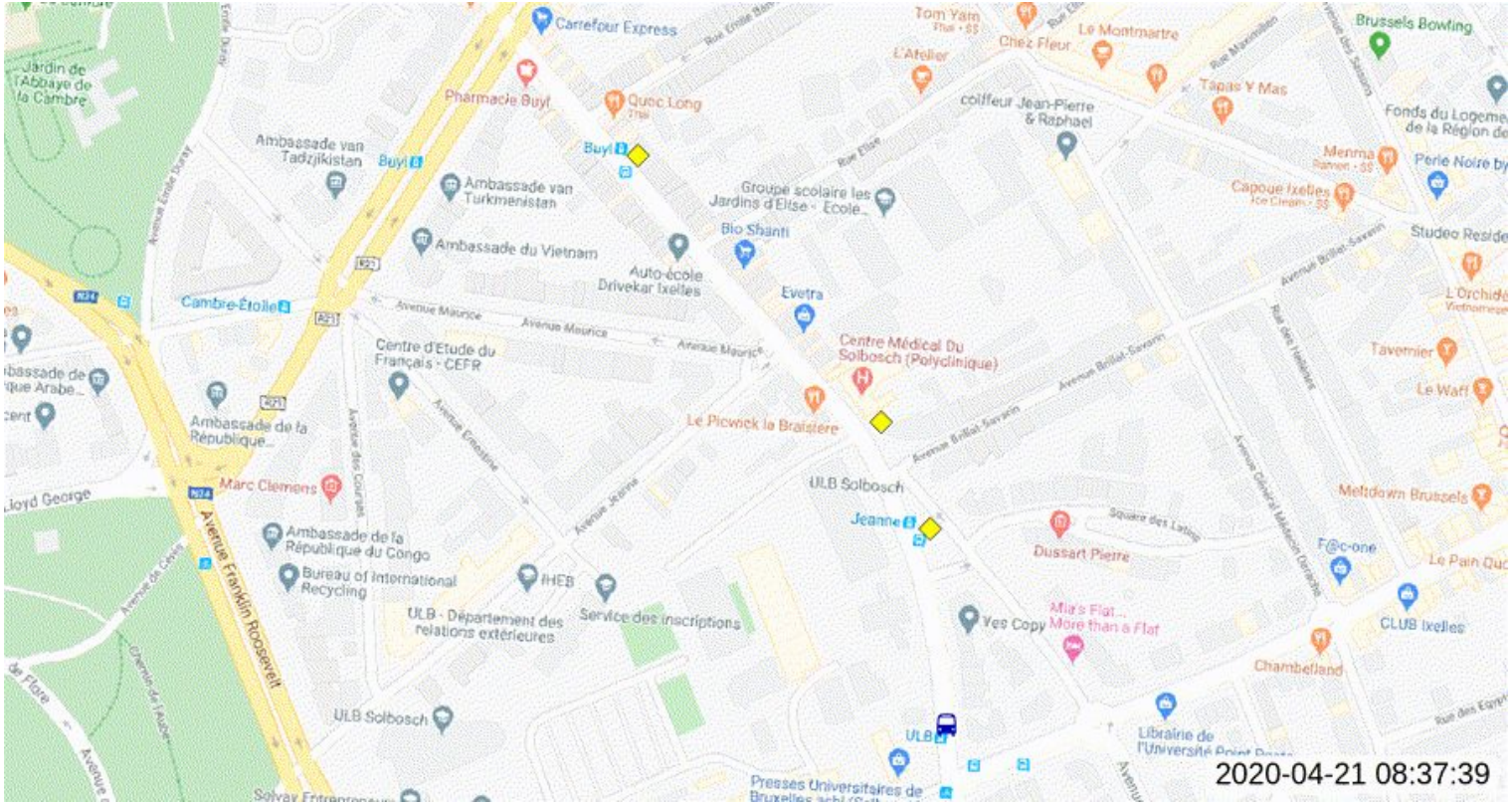
Spatiotemporal Proximity: Marine Data

Spatiotemporal Proximity: COVID-19

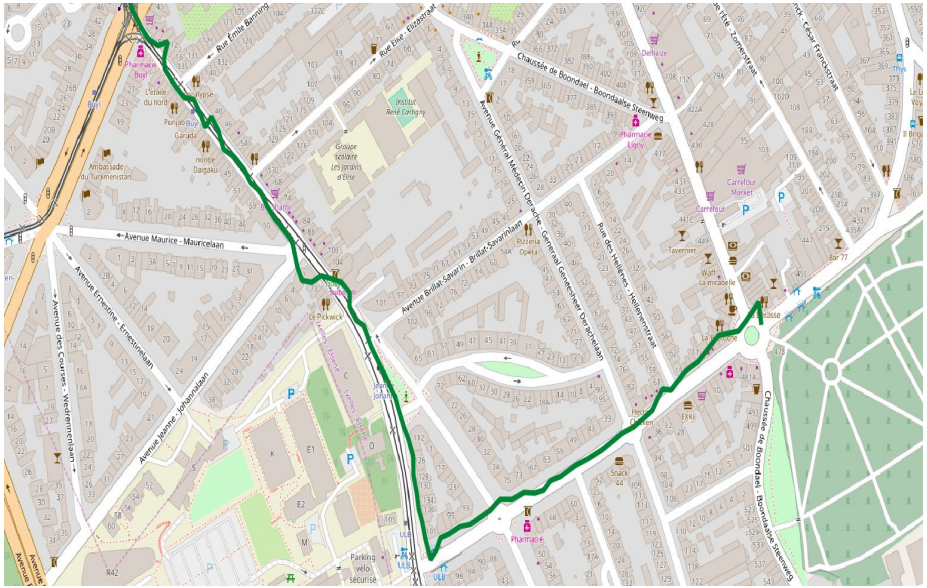


https://ec.europa.eu/info/sites/info/files/recommendation_on_apps_for_contact_tracing_4.pdf

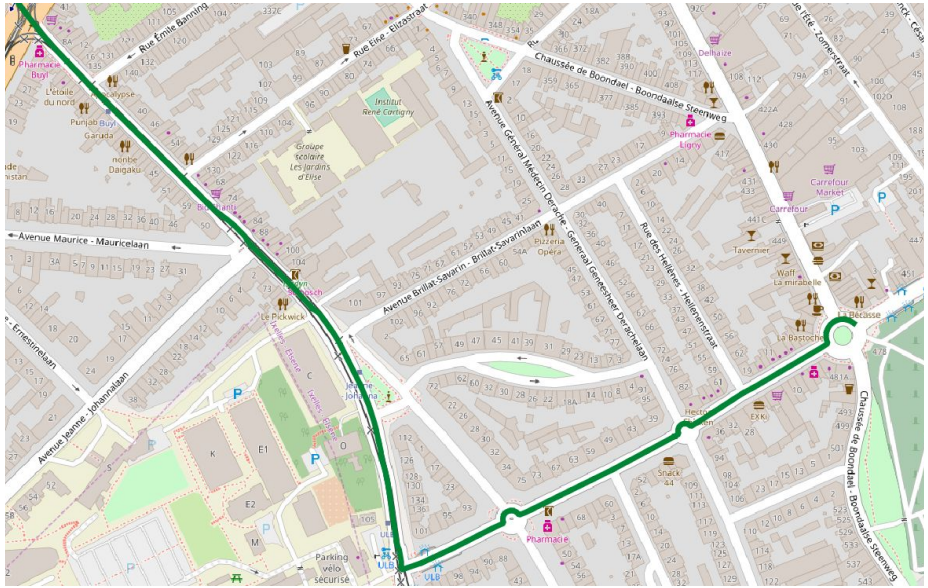
Spatiotemporal Proximity: Smart Advertising



Map Matching - continuous trajectories

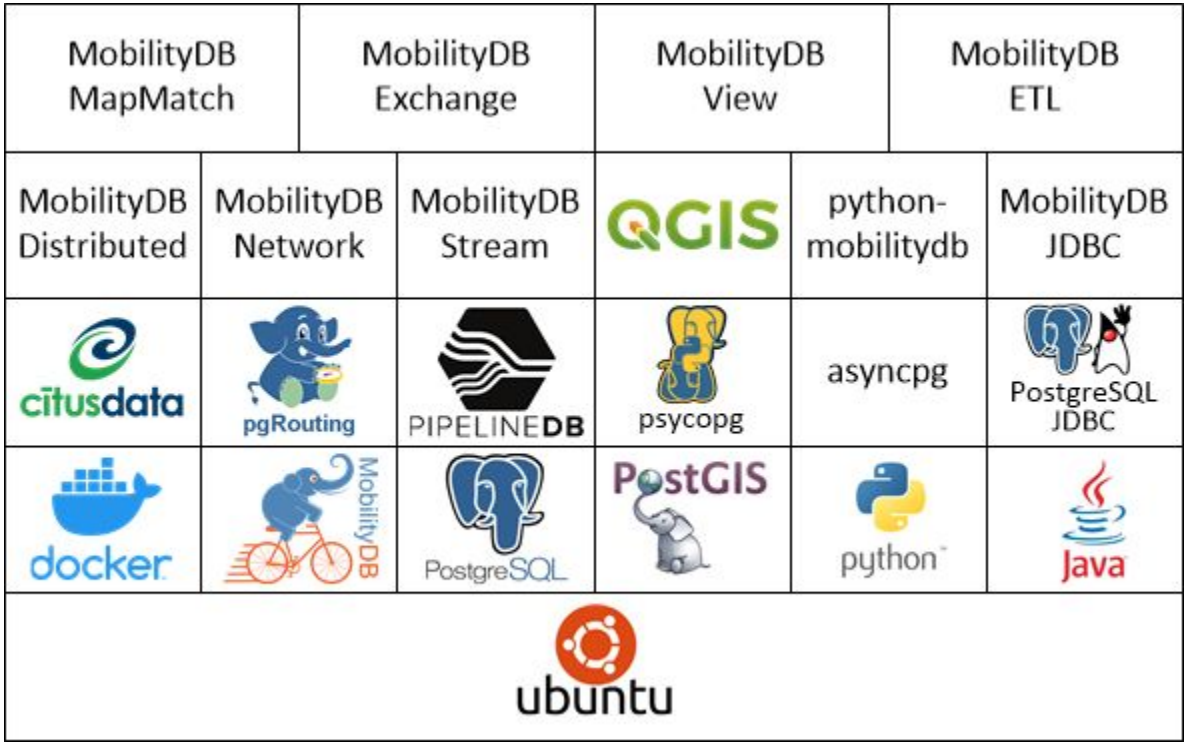


Original



Map-matched

MobilityDB Ecosystem



MobilityDB

An open source geospatial trajectory data management & analysis platform

Location tracking devices, such as GPS, are nowadays widely used in smart phones, and in vehicles. As a result, geospatial trajectory data are currently being collected and used in many application domains. MobilityDB provides the necessary database support for storing and querying such geospatial trajectory data.

MobilityDB is implemented as an extension to PostgreSQL and PostGIS. It implements persistent database types, and query operations for managing geospatial trajectories and their time-varying properties.

A geospatial trajectory is generally collected as a sequence of discrete location points and timestamps, as illustrated in the top most figure. In reality, however, the movement is continuous. Therefore MobilityDB interpolates the movement track between the input observations, as illustrated in the figure in the middle. As such, the moving object location and properties can be queries, effectively approximated, at any time instant. While this interpolation restores the movement continuity, it does not correspond to an increased storage

Moving Features SWG

Chair(s):
Ishimaru, Nobuhiro (Hitachi, Ltd., Defense Systems Division)
Kim, Kyoung-Sook (National Institute of Advanced Industrial Science & Technology (AIST))
SAKR, Mahmoud (Université Libre de Bruxelles (ULB))

Group Charter:
Download Charter document

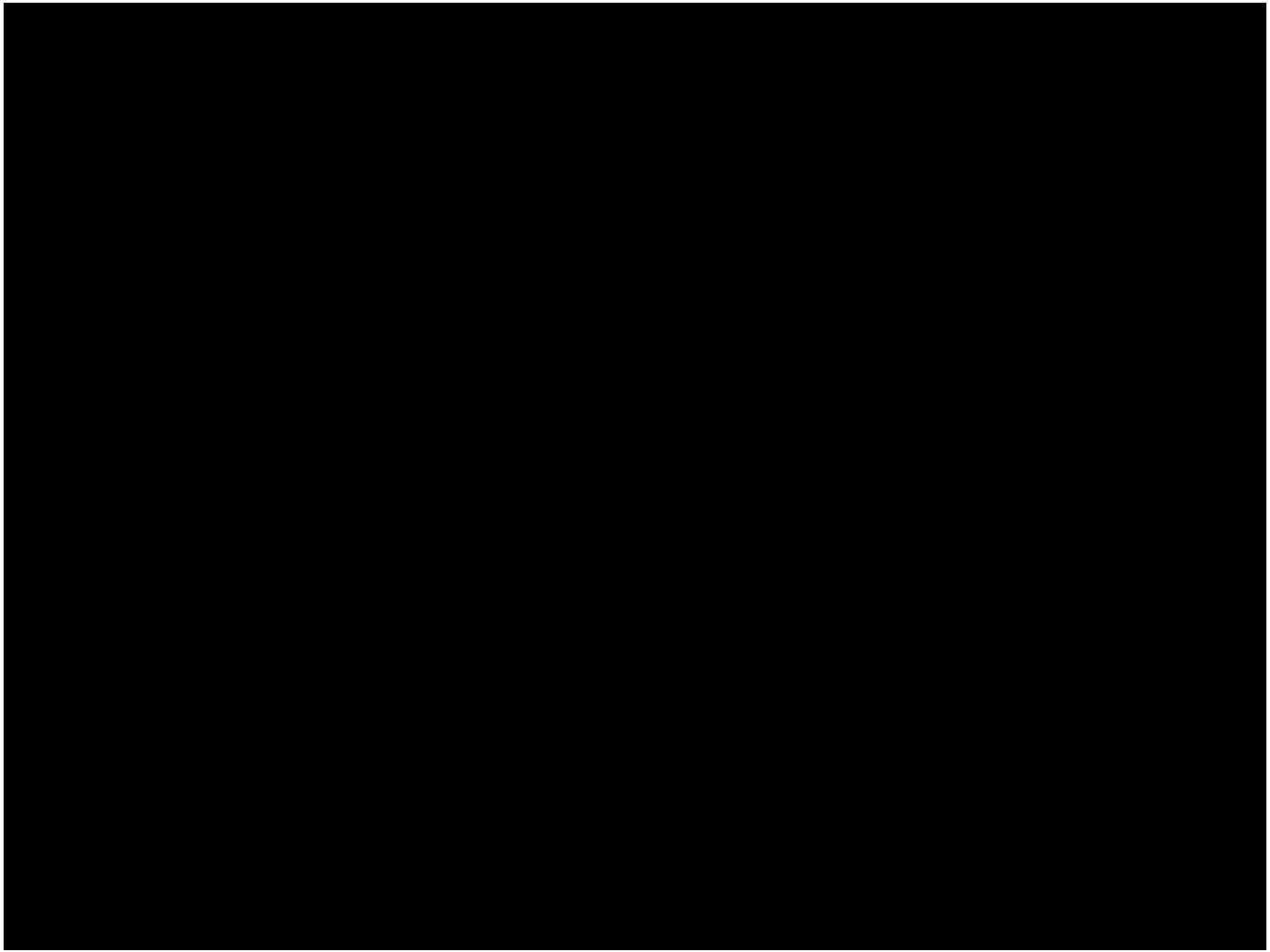
Group Description:

1. Moving Features

With the development of communication and positioning technology such as Global Navigation Satellite System (GNSS), Wi-Fi, and Beacon, collecting movement data for moving features, typically on vehicles and pedestrians, has become easy. A moving feature is a feature whose location continuously changes over time. A moving feature is widely used in several application domains, such as LBS (Location Based Service), marketing, and public health. These applications have considered not only the current location of features but also the historical data of a feature's movement. These data are used to analyze the patterns of moving features and provide input to predictive models. Moreover, innovative applications for smart cities require the overlay and integration of moving feature data from different sources to create enhanced social and business values. For example, sharing moving feature data widely and seamlessly helps organizations to handle marketing at the micro-level, trace people contacts in penedemics, make an efficient evacuation plans in the case of a sudden disaster, control autonomous vehicles and personal mobility, and more based on people's activities and movement conditions.

Functions to Support Visualization in QGIS

```
-- asGeometry(tpoint, bool): {LinestringM, MultiLinestringM}

//good for static visualization
SELECT st_asText(asGeometry(trip, false)) FROM trips LIMIT 1;
-- "LINESTRING M (
      493688.895023848 6580329.05100153 1590995790.978,
      493683.990711951 6580328.07748993 1590995792.478,
      493679.086400055 6580327.10397834 1590995793.378,


//good for animated trajectories
SELECT st_asText(asGeometry(trip, true)) FROM trips LIMIT 1;
-- "MULTILINESTRING M (
     (493688.895023848 6580329.05100153 1590995790.978,
     493683.990711951 6580328.07748993 1590995792.478),
     (493683.990711951 6580328.07748993 1590995792.478,
     493679.086400055 6580327.10397834 1590995793.378),
```

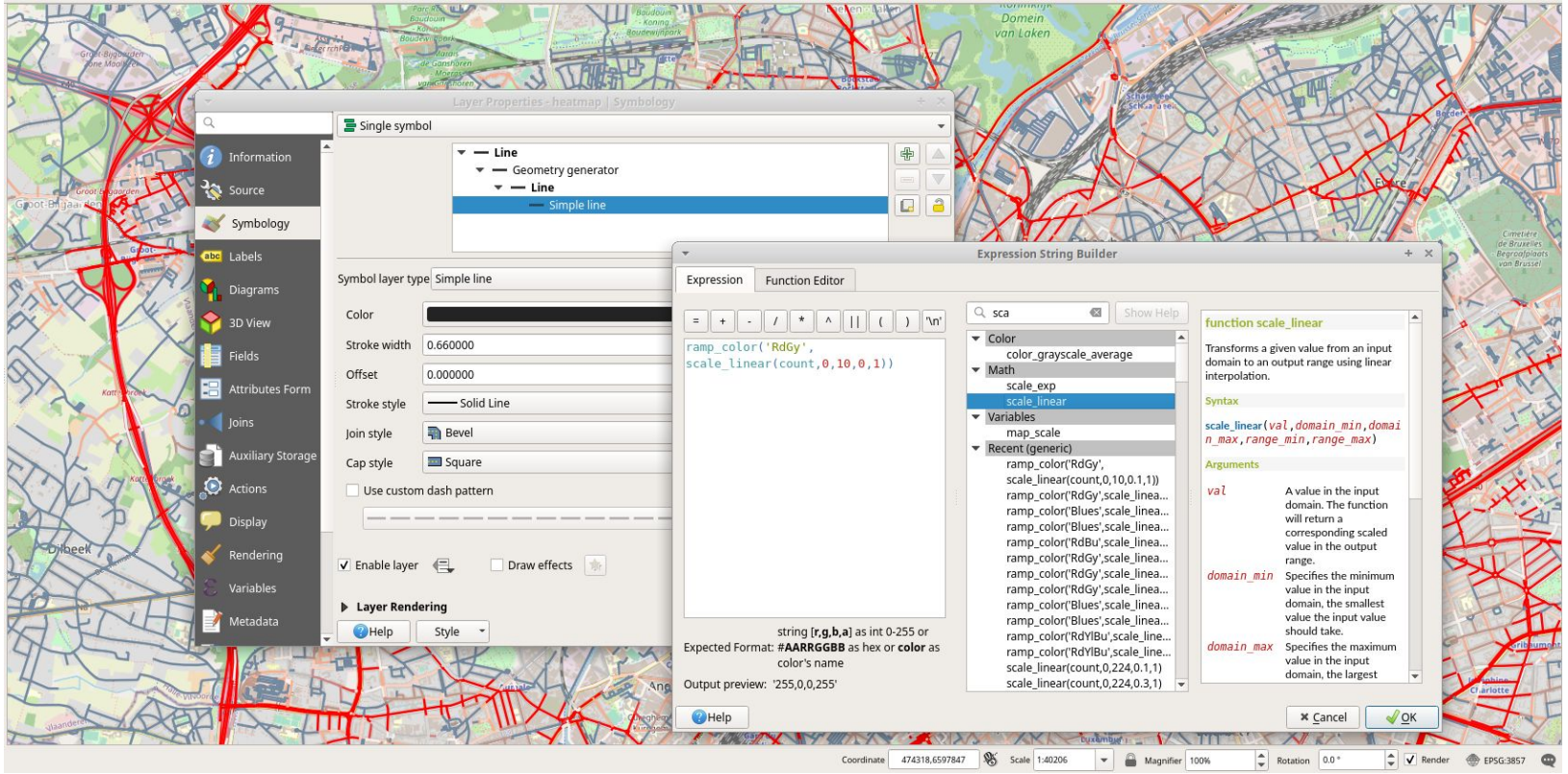


```
Blue linestrings
asGeometry(
   tpoint,
   false)


Red vehicles
asGeometry(
   tpoint,
   true)
  +   Few more tricks
```

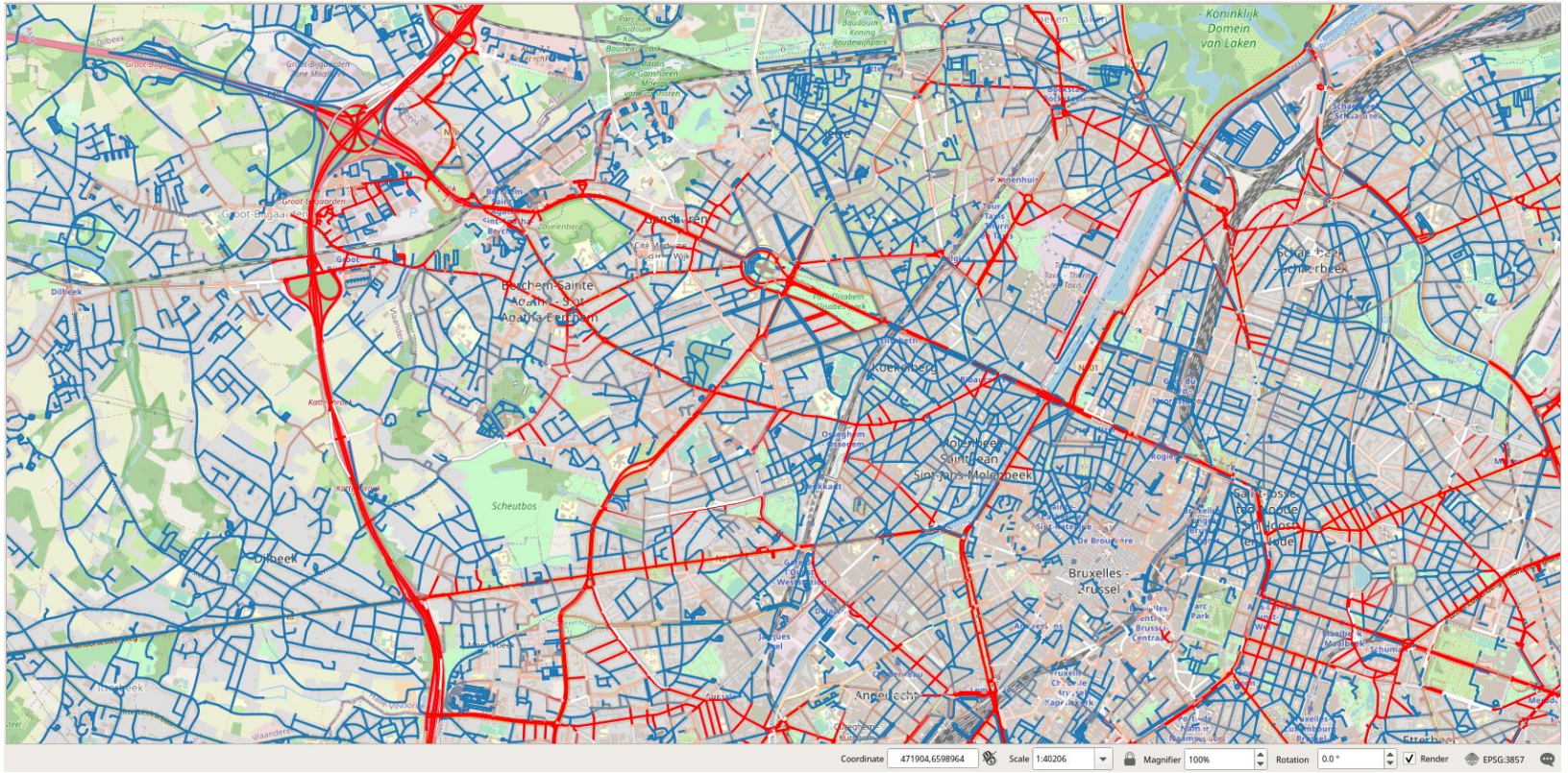
Creating Speed Maps

```
//determine how many trips traversed each of the edges of the network
CREATE TABLE HeatMap AS
SELECT  E.id, E.geom, count(*)
FROM Edges E, Trips T
WHERE st_intersects(E.geom, T.trajectory)
GROUP BY E.id, E.geom;


//add non-traversed edges
INSERT INTO HeatMap
SELECT E.id, E.geom, 0 FROM Edges E WHERE E.id NOT IN (
SELECT id FROM HeatMap );
```

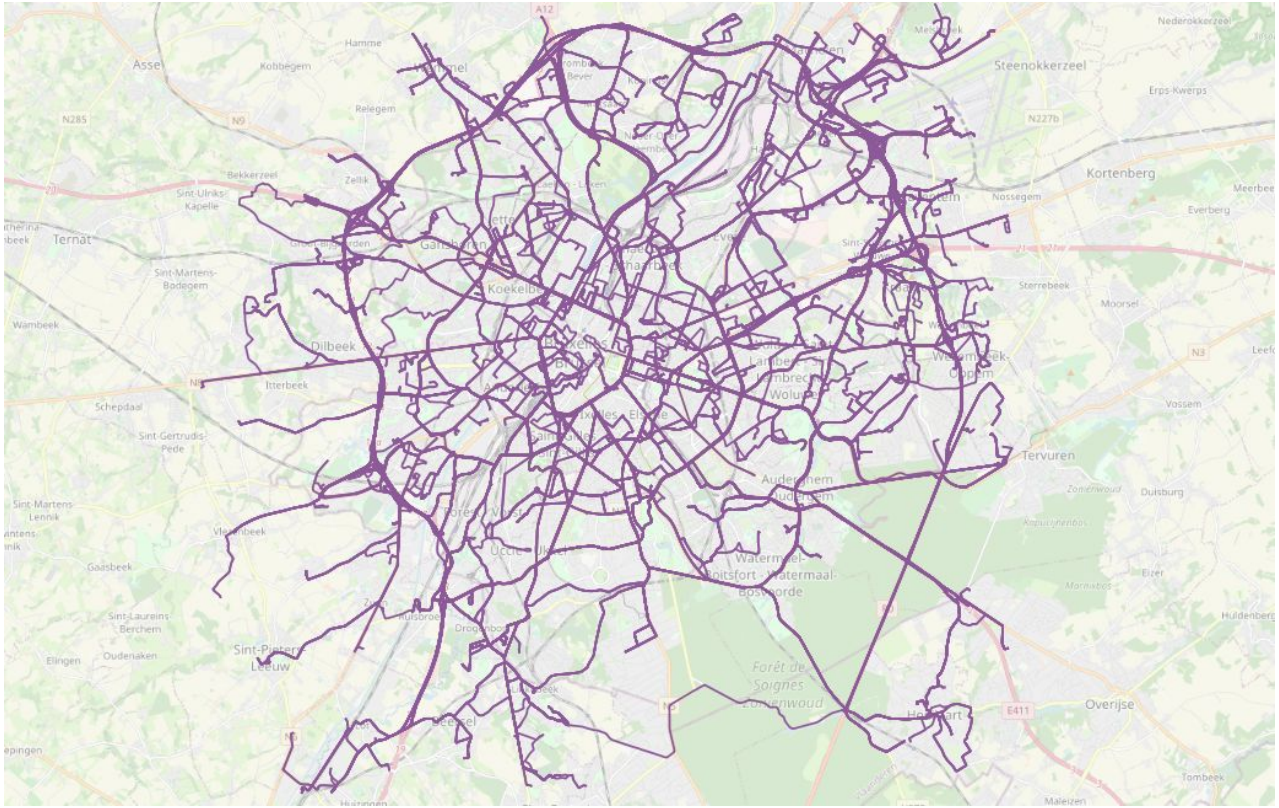
Visualizing Speed Maps

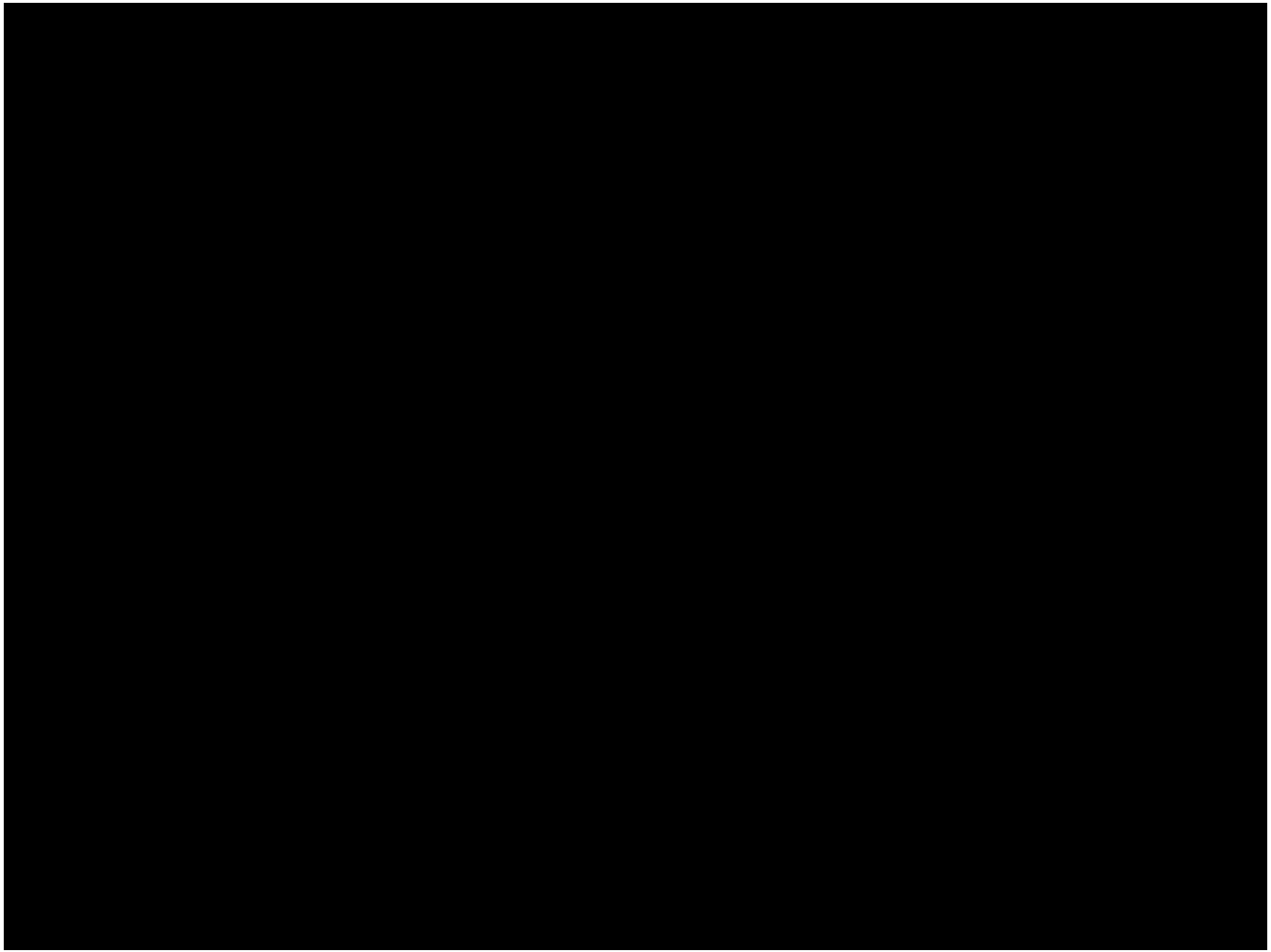


Visualizing Speed Maps



BerlinMOD Data Generator





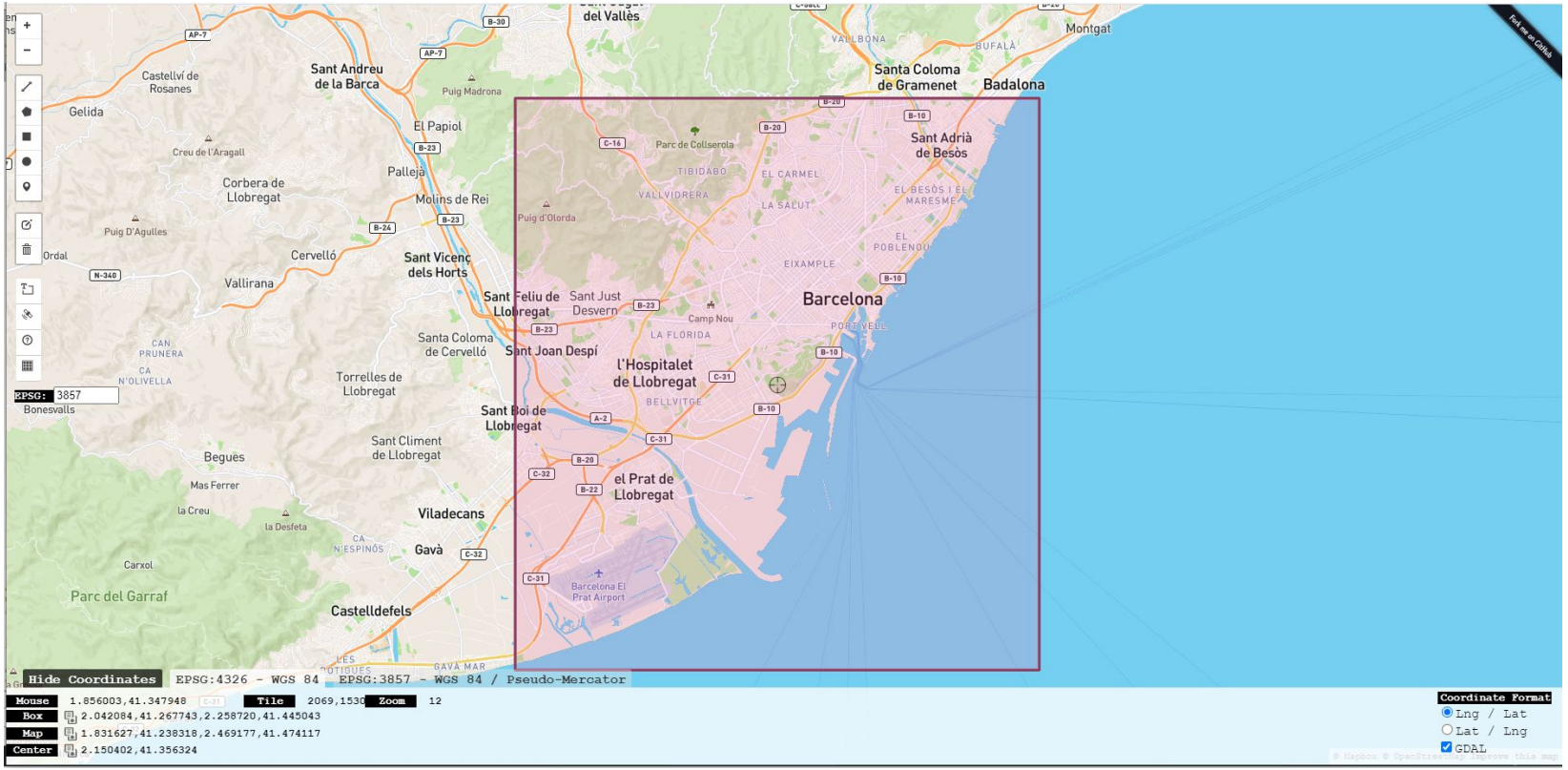
https://drive.google.com/file/d/1lNAdrRqynuCtJE6nJTHAqbI8cgezMxoW/view?usp=sharing

Running The BerlinMOD Generator

1. Download MobilityDB docker image
   https://github.com/ULB-CoDE-WIT/MobilityDB

2. CREATE DATABASE
3. CREATE EXTENSION MobilityDB CASCADE;
4. CREATE EXTENSION pgRouting;
5. Add the street map to the database, PostGIS
6. psql -d brussels -f brussels_preparedata.sql
7. psql -d brussels -f berlinmod_datagenerator_batch.sql
8. psql -d brussels
   -c 'select berlinmod_generate(scaleFactor := 0.005)'

Customizing the Generator to Your City

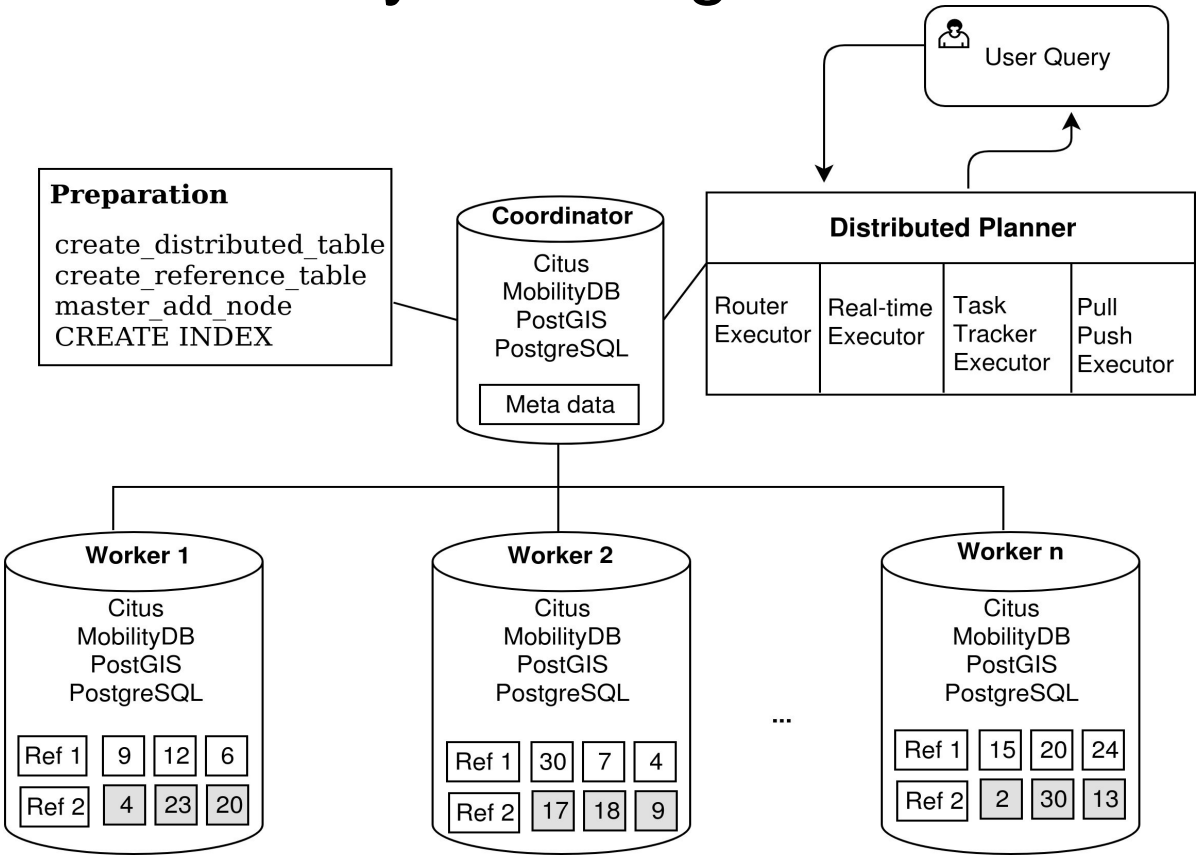


Running The BerlinMOD Generator

```
CREATE EXTENSION MobilityDB CASCADE;
CREATE EXTENSION pgRouting;

CITY="barcelona"
BBOX="2.042084,41.267743,2.258720,41.445043"
wget --progress=dot:mega -O "$CITY.osm"
"http://www.overpass-api.de/api/xapi?*[bbox=${BBOX}][@meta]"

sed -r "s/version=\"[0-9]+\" timestamp=\"[^\"]+\" changeset=\"[0-9]+\"
uid=\"[0-9]+\"
user=\"[^\"]+\"//g" barcelona.osm -i.org

osm2pgrouting -f barcelona.osm --dbname barcelona -c mapconfig_brussels.xml
```

Then run the generator as usual.

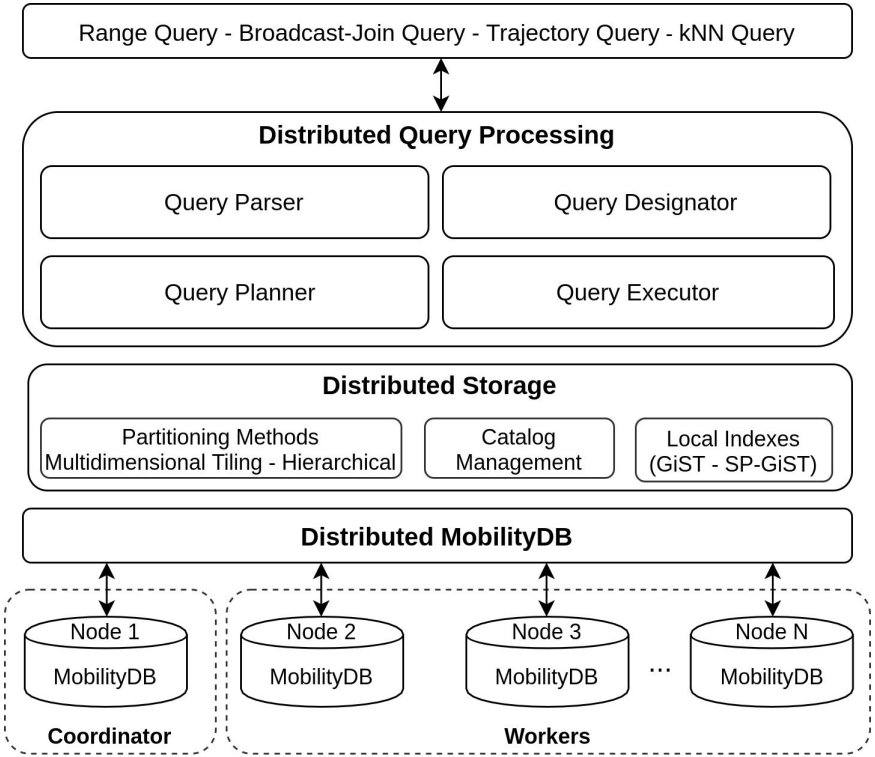
Distributed MobilityDB: Integration with Citus



Distribution: Problem Statement

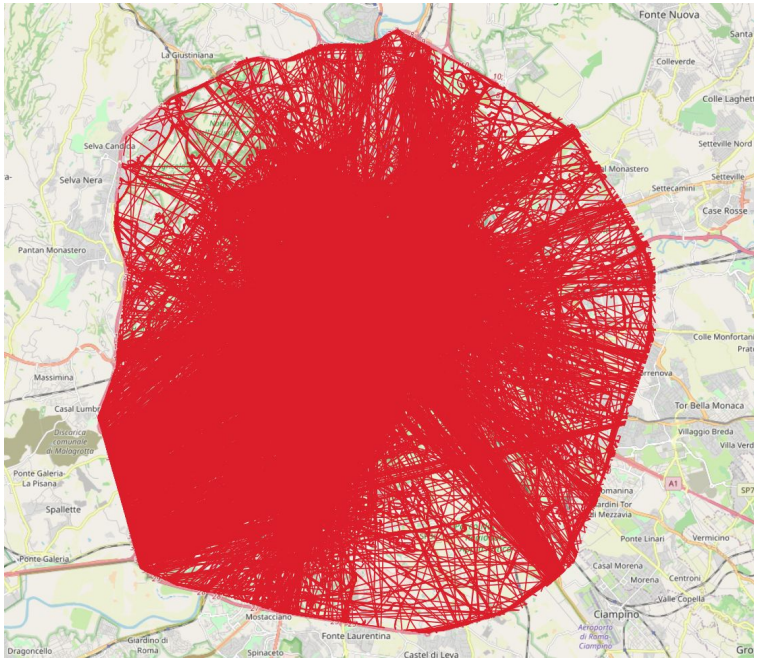
- Citus does not support the spatiotemporal partitioning methods.
- Citus does not understand the MobilityDB operations such as topological functions (e.g., intersects, contains) and geometric/bbox operators (e.g, &&, @>).
- Trajectory data sizes grow quickly and they might contain thousands of spatiotemporal points.
- No other distributed systems that can support trajectory data management in SQL.
- Most of the existing NoSQL systems do not address the problem of continuous trajectories.

Distributed Query Planner



Datasets

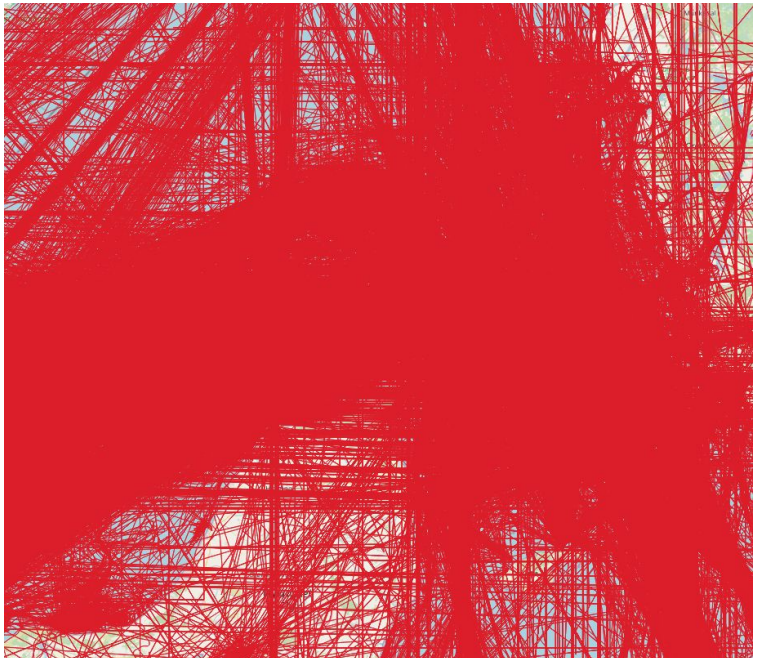
- Roma Taxi Cabs Trajectories:
  - Dataset of mobility traces of taxi cabs in Rome, Italy.
  - It contains more than 16M spatiotemporal points that form 9K trajectories



https://crawdad.org/roma/taxi/20140717/taxicabs/

Datasets

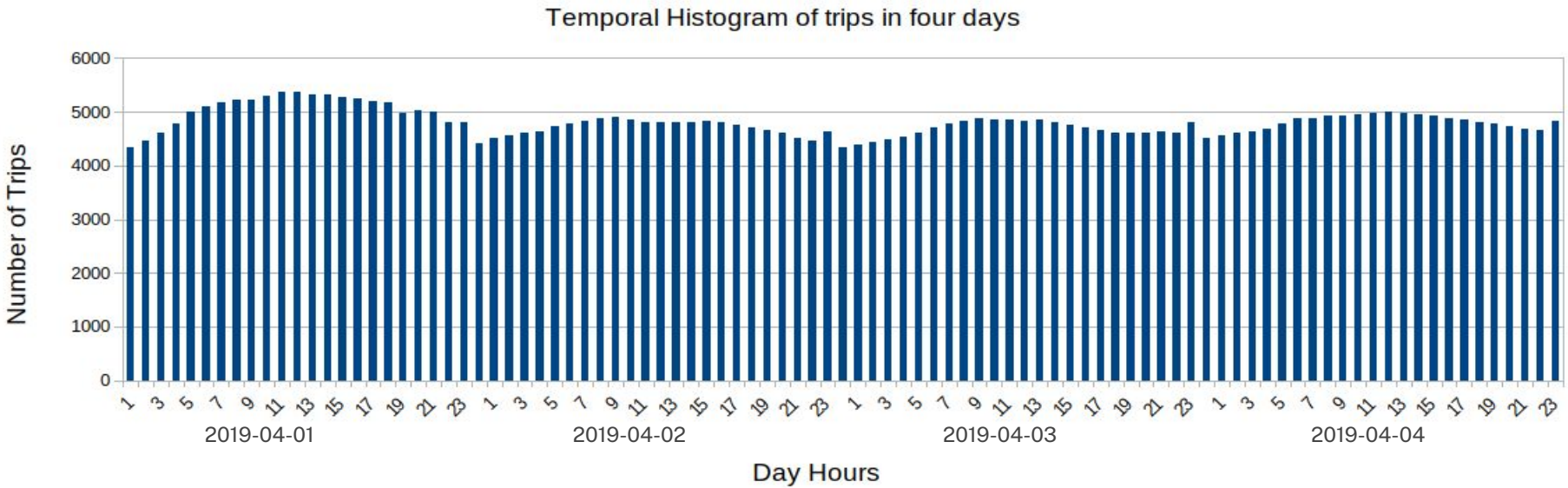
- AIS Trajectories:
  - Real AIS ship trajectory data obtained from the Danish Maritime Authority.
  - It contains more than 10B spatiotemporal points that form more than 400K trajectories.
  - Many of its trajectories contain more than 70K points.



https://www.dma.dk/SikkerhedTilSoes/Sejladsinformation/AIS/Sider/default.aspx

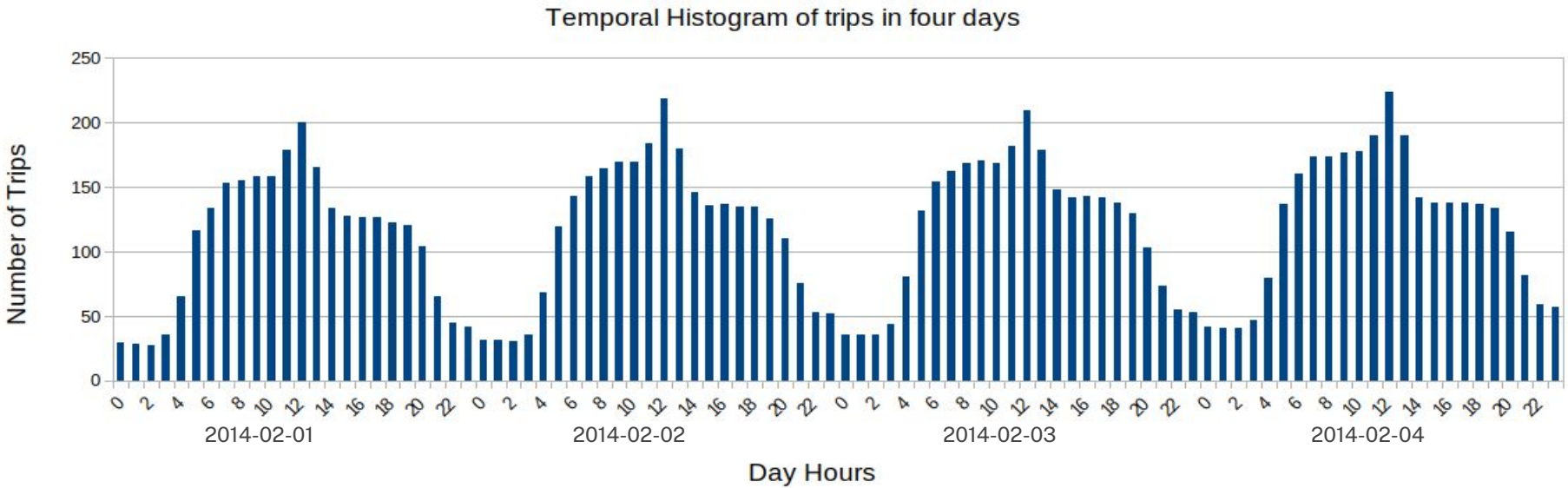
Data Distribution

- Multidimensional Tiling: Real AIS ship trajectories
  - The decision to cut the temporal partitions will cause many overlaps.
  - Less temporal variance



Data Distribution

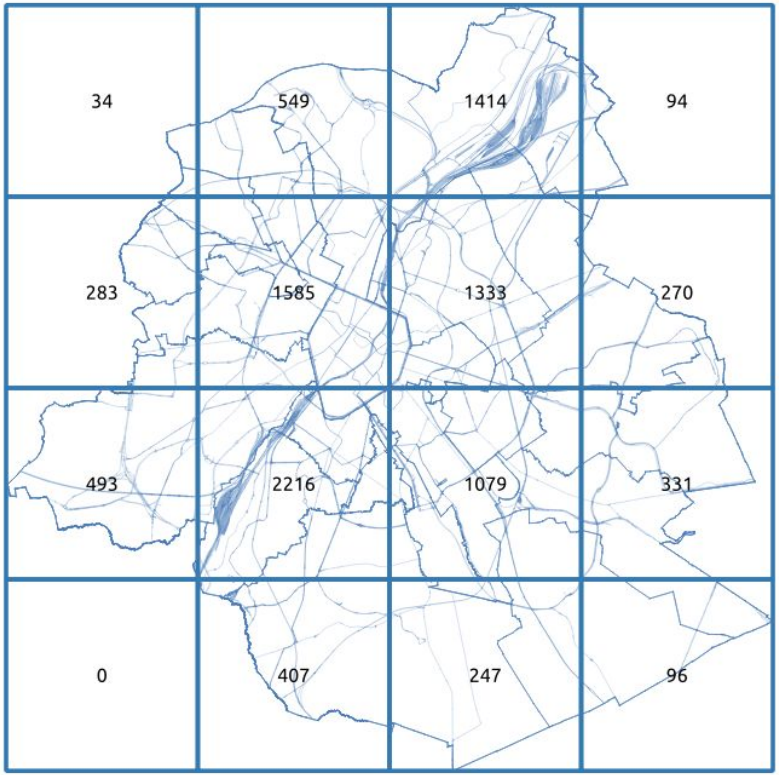
- Hierarchical Partitioning: Taxi datasets
  - No many overlaps
  - Higher temporal variance.



Multidimensional Tiling

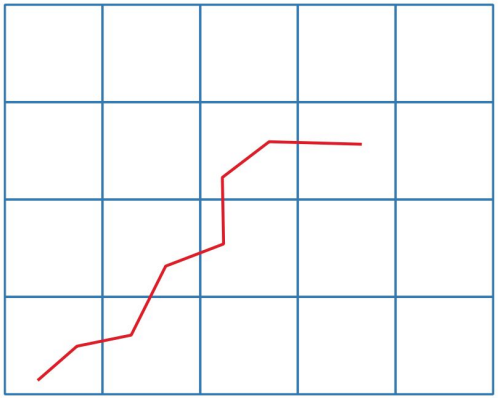
- Fixed-sized tiles
  - Uniform grid partitioning.
  - Tiles are equal in size to each other.
- Count-based tiles
  - OCT-tree partitioning.
  - Tiles are generated according to the distribution density.
- Geometry-based tiles
  - The shape of tiles is based on a given regions table such as provinces, countries, etc.

Multidimensional Tiling: Fixed-sized tiles

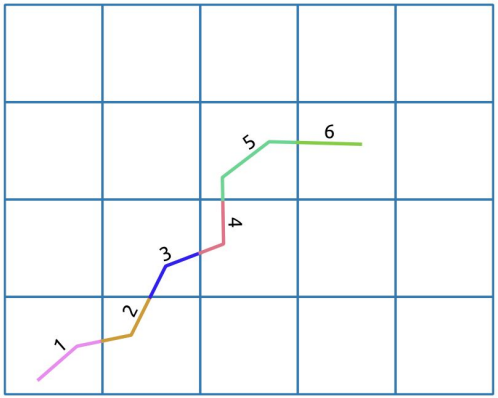


Two options for storing the trajectory

- Non-split partitioning
  - The trajectory overlaps 6 partitions and will be replicated into all of them



- Split partitioning
  - The trajectory overlaps 6 partitions and will be divided into 6 segments.

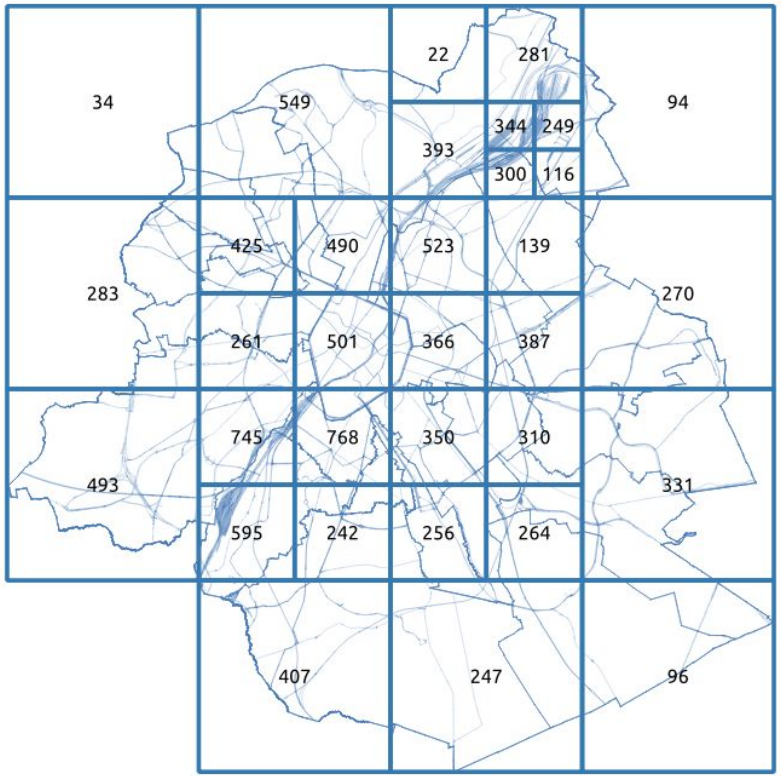


Multidimensional Tiling: Count-based tiles

- **Goal**
  - Spatiotemporal proximity
  - Disjoint partitions
  - Load-balanced partitions
- **Two Phase Execution:**
  - Phase 1:
    - Partition the extent of all trajectories into a number of tiles.
    - Divide the trajectory into small segments.
  - Phase 2:
    - Apply an adaptive grouping method to generate a number of tiles equal to the number of the worker partitions in the cluster.

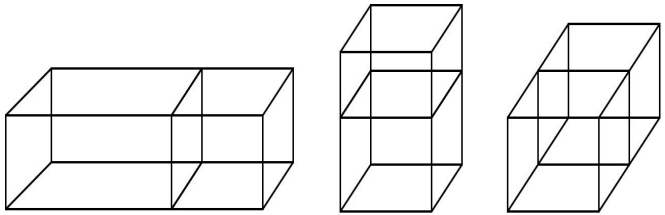
Multidimensional Tiling: Count-based tiles

- **Phase 1**: Tiles Generation
  - Disjoint tiles
  - Threshold on the number of spatiotemporal points inside every tile.
  - Z-order encoding for later grouping in the next phase
  - The trajectory is segmented into a number of segments according to the overlapping tiles, where each segment goes to one tile.

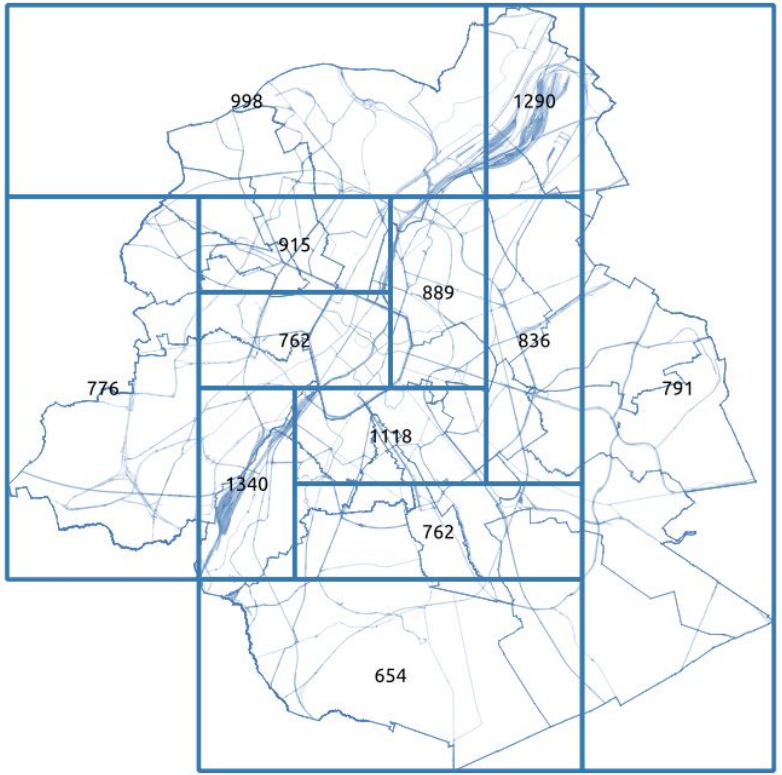


Multidimensional Tiling: Count-based tiles

- **Phase 2**: Adaptive Grouping Method
  - Group tiles that share one face

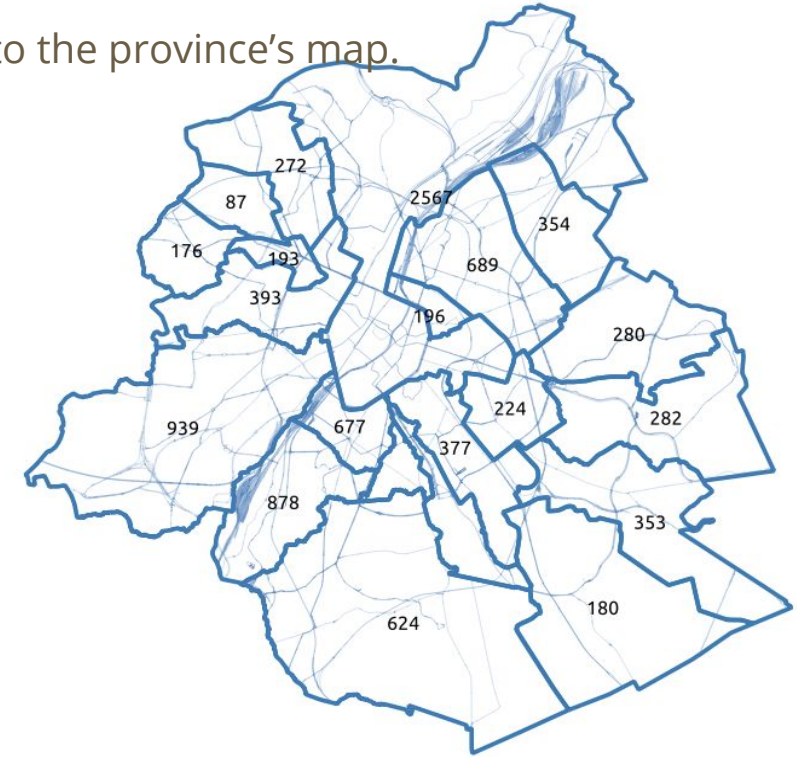


  - The total number of spatiotemporal points in every pair of tiles does not exceed the threshold.



Multidimensional Tiling: Geometry-based tiles

Trajectories are partitioned according to the province's map.



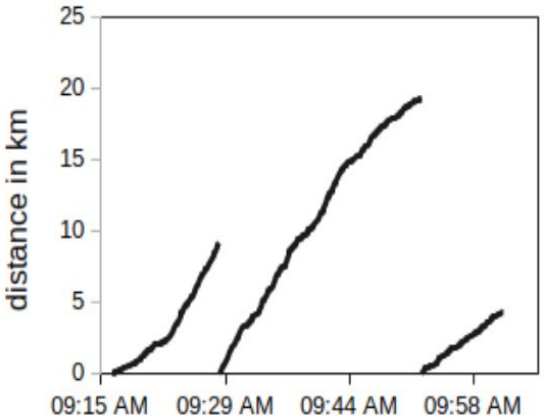
Distributed Conceptual Model

- The distributed function of the query operation has to be defined in terms of **worker-combiner-final**
- In catalogue: worker-combiner-final
  - **Worker** function to perform the function on the local partition
  - **Combiner** function to join the result of two worker functions to produce a single one.
  - **Final** function to produce the final result
- Definition:

```
CREATE DISTFUNCTION functionName(tgeompoint)
(
        WORKERFUNC = function_worker ,
        COMBINERFUNC = function_combiner ,
        FINALFUNC = function_final
);
```

Distributed Conceptual Model: cumulativeLength

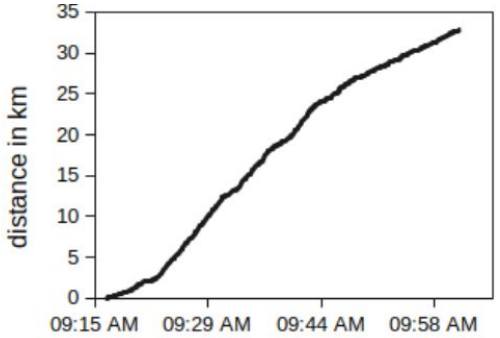
```
CREATE FUNCTION cumulativeLength_worker(traj tgeompoint)
        RETURNS tfloat[] AS $$
DECLARE
        result tfloat[];
BEGIN
        SELECT ARRAY(cumulativeLength(traj)) INTO result;
        RETURN result;
END;
$$ LANGUAGE PLPGSQL IMMUTABLE STRICT;
```



```
CREATE FUNCTION cumulativeLength_combiner(res1 tfloat[], res2 tfloat[])
        RETURNS tfloat[] AS $$
BEGIN
        RETURN array_cat(res1, res2);
END;
$$ LANGUAGE PLPGSQL IMMUTABLE STRICT;
```

Distributed Conceptual Model: cumulativeLength

```
CREATE FUNCTION cumulativeLength_final(partialResults tfloat[])
        RETURNS tfloat AS $$
DECLARE
        cnt integer;
        i integer;
        len float= 0;
        sorted tfloat[];
        merged tfloat[] = '{}';
BEGIN
        SELECT array_sort(partialResults) INTO sorted;
        cnt= array_length(sorted, 1);
        FOR i IN 1..cnt LOOP
                merged= array_append(merged, arr_sort[i] + len);
                len = endValue(merged[i]);
        END LOOP;
        RETURN merge(merged); //convert array into tfloat
END;
$$ LANGUAGE PLPGSQL IMMUTABLE STRICT;
```



Supported Queries: Range Query

Query Text: Find the trips that overlap a given query range

```
SQL:  SELECT tripId FROM trips
      WHERE intersects(trip, 'Polygon((...))') AND
            trip && period '[2012-01-01, 2012-01-05)'
```

```
Plan:      Distributed MobilityDB Plan (Range Query):
            -> Scanning the global index: (Multidimensional Tiling):
             -> Total number of partitions:36
              -> Filter: (shard_bbox && StBOX('Polygon((...))', period '[2012-01-01, 2012-01-05)'))
                  -> Partitions Removed by Filter: 27
            -> Remove Duplicates
             -> Number of Parallel Tasks: 9
              -> Task 1 (WorkerNode1):
                  -> Local Query:
                    -> SELECT tripId FROM trips_shard_3 WHERE intersects(trip, 'Polygon((...))') AND
                         trip && period '[2012-01-01, 2012-01-05)'
                  -> Local Plan:
                    -> Index Scan using trips_shard_3_spgist_idx on trips_shard_3
                    -> Index Cond: (trip && StBOX)
```

Supported Queries: Trajectory Query

Query Text: Find the trips in a given set of ids, if their speed greater than 25 m/s

---

SQL:
```
SELECT tripId, trip FROM trips
WHERE tripId in (1, 2, 10, ...) AND
      speed(trip) ?> 25
```

Plan:
```
Distributed MobilityDB Plan (Trajectory Query):
 -> Scanning the global index: (Multidimensional Tiling):
  -> Total number of partitions: 36
 -> Merge: (trip)
   -> Merge Key: tripId
  -> Number of Parallel Tasks: 36
   -> Task 1 (WorkerNode1):
      -> Local Query:
        -> SELECT tripId, trip FROM trips_shard_4 WHERE tripId in (1,2,10) AND speed(trip) ?> 25
      -> Local Plan:
        -> Bitmap Heap Scan on trips_shard_4
        -> Filter: (speed(trip) ?> 25)
        -> Bitmap Index Scan on trips_shard_4_triptId_btree_idx on trips_shard_4
        -> Index Cond: (tripId = ANY ({1,2,10}))
```

Thanks for listening !