# 5 Case Study: Lockdown traffic speeds

In this case study we explore the possibilities and limitations of vector tiles to accommodate the temporal density of a dataset originally published as a live stream. For this purpose we chose to visualize changes in traffic speeds in the city of Brno. The temporal range of the source dataset e period form the 16th of March to the 10th of May 2020[[1]](#footnote-21). This time period coincidentally matches with the first period of government restrictions in Czech Republic to prevent the spread of the COVID-19 pandemic. The size, spatial and temporal detail of the dataset posed a challenge both in terms of data processing as well as in terms of designing the interactive cartographic visualisation[[2]](#footnote-22).

## 5.1 Data sources and transformations

The source raw data were formatted as (compressed) CSV files containing an estimate traffic speeds at a specific location, at a specific time, based on historical observations. One such file showed the expected traffic speeds during one week[[3]](#footnote-24).

Spatially, one file covered the area of a zoom level 6 tile, which meant that the data files for our problem area also covered a significant part of the Czech Republic (see Fig).



**Fig.** There are four tiles at zoom level six that cover Czech Republic, our area of interest, Brno municipal region fits into tile 120212 (screenshot taken from <https://labs.mapbox.com/what-the-tile/>).

As for the original CSV structure, one line in the file represents one road segment. Each segment was identified by a pair of OpenStreetMap node IDs representing a start node and the end node of a road segment. Note that the node ordering also determines the direction of the recorded traffic, which means that bidirectional routes were recorded twice in the file – one row for direction from node A to node B and another row for speeds in direction from B to A. Following the two columns containing node identifiers, there were 2016 columns containing speed estimates in 5 minute intervals per each segment (7 days × 24 hours × 12 five-minute periods). An example row in a CSV file could look like: *113054533,113096757,54,54,…57*, where the first two digits are node identifiers followed by an array of traffic speeds. All speeds were recorded in kilometers per hour. The starting speed record corresponds with Sunday 00:00 AM of the given week in the files time zone. The records continue in 5 minute increments until the concluding record marking the end of the week.

This gives us an idea of the data volumes that needed to be processed. One file for the zoom level 6 tile (see fig.) contained approximately 1 086 958 lines representing the line segments (the line count could differ across the files as speeds were not provided for segments for which the volume and quality of data did not allow a high confidence estimate). Each row contained 2018 records (speeds + identifiers) which exceeds the default maximum column count per table in a PostgreSQL database (250 – 1600 based on column type)[[4]](#footnote-28). Data was provided for eight weeks, so there were eight files files of these proportions to be processed.

There were several tasks to be completed in the initial phase of data processing. As the OpenStreetMap node IDs do not directly contain the spatial information, the actual coordinates for each node needed to be obtained. This was done in the following steps. First, to minimize redundant API calls later, we extracted the unique node IDs form the first two columns. As we have seen earlier, the node IDs can appear several times as route identifiers, either in bi-directional segments or in crossroads and other structures[[5]](#footnote-30). For each of the unique nodes the spatial coordinates were obtained by querying the Open Street Map API[[6]](#footnote-32).

With spatially defined unique nodes it was possible to filter out the subset of the nodes that belonged to the Brno municipal area. The most straightforward way to do that was to load the nodes to QGIS desktop to perform *select by location* against the polygon of the city area (with five kilometer buffer to provide some context of immediate surroundings). Armed with a collection of Brno nodes (the count was 131 257), we returned to the original traffic speed CSVs to extract the nodes from Brno, this time with speed attributes. The challenge was in searching for 131 257 nodes in the superset of 1 086 958 lines and then extracting the matching lines, each with all of its 2018 attributes[[7]](#footnote-34).

Such task is reminiscent of situations described in the *Small big data manifesto* (Voss, Lvov, & Lewis (2012)) – even though the big data is mainly associated with large scale clustered infrastructure, individuals increasingly come across situations when they need to process large dataset only with a single machine at their hands. Setting up a cluster of machines is not viable for many applications due to financial, time or skillset demands. For one-time processing of data that does not fit into memory, we are left with a range of simple but often efficient computing tools and approaches (Turner-Trauring (2020)). One of them is reading input data in chunks that can fit to memory, applying a processing function to these chunks and using a reducer function that can combine the processed chunks into a final result. This way the memory size limitation is bypassed, however, computation time of the processing function can still become a bottleneck. Multi-threaded execution can ease the problem by running the execution function in parallel on individual CPU cores. The size of chucks and the number of threads needs to be fine tuned to fit the capabilities of given hardware, but in general these techniques can significantly reduce the processing time even on modest machines. Cycling back to our speed files, a simple script combining chunking and parallelization (using the Dask Python library) was able to complete the extraction of Brno segments from one week file in 3 min 32.8s (on Intel i7 8 cores, 30 GiB RAM).

The output of the previous operation was a list of eight CSV files in the original structure showing the estimated speeds for road segments in Brno. These weekly files where split into smaller chunks representing individual days to avoid hitting the database column length limitations[[8]](#footnote-36). The resulting set of 56 files with 288 columns of speed data were finally loaded to the PostgreSQL database. At this point, the tables of Brno node pairs and node coordinates were also imported in order to create a line segment layer from the point coordinates using PostGIS plugin[[9]](#footnote-38). From now on, the daily speed tables could be joined with the table of line segments to create futures spatial layers[[10]](#footnote-40). During this process various visualisation experiments have been done using QGIS connected to the database. As a result of these experiments a decision has been made to reduce the temporal granularity of the speed layers from 5 minute intervals to one hour averages[[11]](#footnote-42). This significantly reduce the storage overhead in generated vector tiles while maintaining sufficient information density for visualisation purposes.

A database loaded with road spatial layers with associated hourly speed attributes provides a solid starting point from which many avenues could be taken, either in analytical or visualisation direction. Our focus is on interactive cartographic visualisation with vector tiles, therefore we created the necessary amount of vector tiles from GeoJSON exports from the database using the tippecanoe command line tool[[12]](#footnote-44). The batch of resulting *.mbtile* files was then uploaded to the Mapbox server via API[[13]](#footnote-46).

## 5.2 app architecture

The building blocks of the application are basically the same as with the case study described in the previous chapter. Even though the PostgreSQL database played a vital role in the data preparation phase, the final application does not use it for back-end data storage. Instead, the vector tiles have been uploaded to Mapbox to act as a vector tile server. The front-end application is build using React and Redux for state management, mapbox-gl.js is used as a rendering engine on the client.

## 4.3 Cartographic decisions

Two ways of representing time: – repr. time with space (e.g. time lines), rep. time with time (animation)

Aim – all data shown from zoomlevel 10 (compare with mapbox default layer)

zoom based parameters – road width and offset (screnshots from the studio)

comparison – options, why 3D was selected

## 4.4 User interface design

TODO: note on scale levels in mapbox streeets layer (example with live traffic screening)

https://api.mapbox.com/styles/v1/ppeettoo/ck4yfkusp1ejb1cmnpfnwvecc.html?fresh=true&title=view&access\_token=pk.eyJ1IjoicHBlZXR0b28iLCJhIjoiY2loN21nMTBuMHQ1YXVta2l6YjJzbHM4YSJ9.cpjhFrPRqzOF037dktUgBw#11.06/49.1846/16.6589

Images: img-live-mb-traffic-1,2,3.png

## 4.5 Evaluation and possible extensions

* what spatio-temporal queries are enabled by this kind of visualisation? Which are not? (see chapter 2)

<https://github.com/pondrejk/dizzer/blob/master/misc/scripts/06-run_length_encode.py>

draft classification:

* high average speeds, low variability (highways)
* low speeds and variability (tiny segments)
* variable speeds throughout the day (inner city alleys)

^ how this all changed during the lockdown?

caveats Traffic speed does not bear information on car density? – what is the relation? in pandemic it should be higher?

Exceed the tile size and (tippecanoe) and self host?

The visual analysis tool should work equally well regardless of the velocity of data generation or the cadence of change. For that matter, the temporally dense dataset should serve well for designing a cartographic interface even though the dataset is not itself consumed “real-time”.

Turner-Trauring, I. (2020). Process large datasets without running out of memory. *Available online at https://pythonspeed.com/memory/ (last accessed October 26, 2020)*.

Voss, A., Lvov, I., & Lewis, J. (2012). The small big data manifesto. *Available online at https://smallbigdata.github.io/manifesto.html (last accessed October 26, 2020)*.

1. The author would like to thank Mapbox, Inc. for generously providing the traffic data sample for the purpose of this case study. [↑](#footnote-ref-21)
2. Live demo of the application is accessible at <pondrejk.eu/traffic>, screenshots of the interface can be found in appendix c. [↑](#footnote-ref-22)
3. The official description of the data source can be found at <https://docs.mapbox.com/traffic-data/overview/data/> [↑](#footnote-ref-24)
4. Just for completion, the column limit can be extended, but this requires re-compiling the database from the source code. See <https://www.postgresql.org/docs/current/limits.html> for the overview of PostgreSQL limits [↑](#footnote-ref-28)
5. The python script based on the *numpy* library that was written to perform the unique node extraction can be found at <https://github.com/pondrejk/dizzer/blob/master/misc/scripts/01-get_unique_nodes.py> [↑](#footnote-ref-30)
6. The script to do that using the *osm* Python library is available at <https://github.com/pondrejk/dizzer/blob/master/misc/scripts/02-get_node_coordinates.py> [↑](#footnote-ref-32)
7. The script to perform this action (using the *dask* Python library) is available at <https://github.com/pondrejk/dizzer/blob/master/misc/scripts/03-select_segments.py> [↑](#footnote-ref-34)
8. Using this Python script <https://github.com/pondrejk/dizzer/blob/master/misc/scripts/04-split_by_day.py> [↑](#footnote-ref-36)
9. The query using PostGIS’s ST\_MAKELINE available at <https://github.com/pondrejk/dizzer/blob/master/misc/queries/01-create_lines> [↑](#footnote-ref-38)
10. Example query at <https://github.com/pondrejk/dizzer/blob/master/misc/queries/03-streets_join> [↑](#footnote-ref-40)
11. Example query at <https://github.com/pondrejk/dizzer/blob/master/misc/queries/02-generate_hourly_averages> [↑](#footnote-ref-42)
12. <https://github.com/mapbox/tippecanoe> [↑](#footnote-ref-44)
13. The batch upload script is available at <https://github.com/pondrejk/dizzer/blob/master/misc/scripts/05-mapbox_upload.py> [↑](#footnote-ref-46)