5 Case study: 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 between the 16th of March to the 10th of May 2020[1]. The time range coincidentally matches with the first period of government restrictions in Czech Republic to prevent the spread of the COVID-19 pandemic. The 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].

5.1 Data sources and transformations

The source raw data were formatted as (compressed) CSV files containing estimate traffic speeds for specific road segments, at a specific time, based on historical observations. In terms of temporal coverage, one file contained the expected traffic speeds spanning across one week[3]. 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 1).

---

1 The author would like to thank Mapbox, Inc. for generously providing the traffic data sample for the purpose of this case study.

2 Live demo of the application is accessible at <pondrejk.eu/traffic>, screenshots of the interface can be found in Appendix C.

3 The official description of the data source can be found at https://docs.mapbox.com/traffic-data/overview/data/

Fig.1 Czech Republic is covered by four tiles at zoom level six. Our area of interest, the 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 corresponded with one road segment. Each segment was identified by a pair of *Open Street Map node IDs* representing a start node and the end node of a segment. Notice that the node ordering also determined the direction of the recorded traffic. Owing to that, bidirectional routes were recorded twice in the file — one row for direction from node A to B and another row for the opposite direction.

After the two columns with node identifiers, each row contained 2016 more columns with speed estimates in 5 minute intervals per each segment (7 days × 24 hours × 12 five-minute periods in hour). An example of how a row could look like: *113054533*,*113096757*,*54*,*54*,…*57*, where the first two digits were node identifiers followed by an array of traffic speeds. All speeds were recorded in kilometers per hour. The first record represents Sunday 00:00 AM of the given week in the time zone of the mapped area. The records continue in 5

minute increments until the concluding one marking the end of the week.

This gives us an idea of the data volumes that needed to be processed. One file with week-long data per zoom level 6 tile contained approximately 1 086 958 lines representing road segments (the line count could differ across the files because segments lacking sufficient data for a high confidence estimate were omitted). Each row contained 2018 records (speeds + identifiers) which exceeds the limit for the maximum column count per table in a PostgreSQL database (250 — 1600 based on column type)[4]. Data was provided for eight weeks, so there were eight files files of these proportions to be processed.

A number of tasks were completed in the initial phase of data processing. As the OSM node IDs as such do not contain the spatial information, we needed to first obtain the actual coordinates for each node. This was done in the following steps. First, to minimize redundant API calls, we extracted the unique node IDs form the first two columns of our data files. As we have seen earlier, the node IDs can appear several times as route identifiers, either in bidirectional segments or in crossroads[5]. For each of the unique nodes, we obtained spatial coordinates by querying the Open Street Map API[6].

---

4 Just for completeness, 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

5 The python script based on the *numpy* library that performed the unique node extraction can be found at https://github.com/pondrejk/dizzer/blob/master/misc/scripts/01-get_unique_nodes.py

6 The script to do that using the *osm* Python library is available at https://github.com/pondrejk/dizzer/blob/master/misc/scripts/02-

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 is 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 city's road segments, with speed attributes included. The challenge was in searching for 131 257 nodes in the superset of 1 086 958 lines and then extracting the matching lines in full length of 2018 attributes[7].

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 data center infrastructure, individuals increasingly come across situations when they need to process large datasets only with a single machine at their hands. Setting up a cluster of machines is not viable for many applications be it for lacking time, expertise or finances. 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 them and then using a reducer function to combine the processed chunks into a final result. This way the memory size limitation is bypassed, but the computation time of the processing function can still be a bottleneck. Multi-threaded execution can ease the problem by running the function on several CPU cores in parallel. The

---

get_node_coordinates.py

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

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 extraction process yielded eight CSV files in the original structure showing the estimated speeds for road segments within Brno. These weekly files where split into smaller chunks representing daily speeds to avoid hitting the database column length limit[8]. The resulting set of 56 files, each with 288 columns of speed data, was 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 the PostGIS plugin[9]. From now on, the daily speed tables could be joined with the table of line segments to create spatial layers[10]. During this process various visualisation experiments were done using QGIS connected to the database. In light of these experiments, we decided to reduce the temporal granularity of the speed layers from 5 minute intervals to one hour averages[11].

8 Using this Python script https://github.com/pondrejk/dizzer/blob/master/misc/scripts/04-split_by_day.py

9 The query using PostGIS's ST_MAKELINE is available at https://github.com/pondrejk/dizzer/blob/master/misc/queries/01-create_lines

10 Example query at https://github.com/pondrejk/dizzer/blob/master/misc/queries/03-streets_join

11 Example query at https://github.com/pondrejk/dizzer/blob/master/misc/queries/02-

This significantly reduced 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. We decided to explore the vector tile format and its aptness for powering interactive cartographic visualisations. Therefore we created the necessary amount of vector tiles from GeoJSON database exports using the tippecanoe command line tool[12]. The batch of resulting *.mbtile* files was then uploaded to the Mapbox server via API[13].

5.2 Application 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 are loaded from Mapbox tile server. The front-end interface is build using React with Redux for state management, mapbox-gl.js is used as a rendering engine on the client.

---

generate_hourly_averages

12 https://github.com/mapbox/tippecanoe

13 The batch upload script is available at https://github.com/pondrejk/dizzer/blob/master/misc/scripts/05-mapbox_upload.py

5.3 Cartographic decisions

The main requirement for the map was the ability to display the whole dataset at the zoom level 10 so that the municipal traffic network can be observed as a whole. The Mapbox tile server enforces a size limit of 500 KB per tile for the uploaded vector tiles. While this is a reasonable limitation to ensure fast rendering, it is hard to adhere to it with denser datasets especially at smaller zoom levels where individual tiles cover larger area. One way to work around this is by limiting the number of features displayed across scales (see Fig 2), which obviously has a downsize in loosing some resolution of the visualized data.

Fig. 2 Live Mapbox traffic data layer at three zoom levels (city of Brno, largest scale on the left). While reducing the number of displayed roads per importance is a way to maintain fast tile rendering, in case of Brno it precludes observing the city's traffic network as a whole. Styling in Mapbox Studio by the author.

Another way to grapple with this problem without abandoning the Mapbox infrastructure is to break up the tile layers into sub-layers with smaller attribute count. Using this approach we achieved full spatial resolution at the zoom level 10 — we halved the tile layers with daily speed coverage, so that each tile layer covered twelve hours. This got us below the tile size limit but also impacted to the smoothness of the user experience on the client.

When displaying the layer with a data driven style, any user-induced changes to displayed attribute are rendered smoothly when the attribute change is within the same

layer. Once a different layer needs to be loaded, there always is a visible gap between hiding the previously displayed layer and enabling the new one, which unfortunately can not be treated by any ease-in effect in mapbox-gl. One way to work around this is in keeping all the layers in the visible state and change their order dynamically so that the selected layer is on top — however our testing proved this approach inefficient for large number of layers. In our case it was two layers per day, so 2 x 7 (days) x 8 (weeks) = 112 layers.

The color scale selected to visualize traffic speeds spans from 0 to 140 kilometers per hour in 10 kilometer intervals. The color selection was guided by the need for sufficient contrast on the dark background. The break between the two hues used in the color scale rests at 60 km which should ensure the variability of speeds within the slower inner-city routes is visible while the distinction from fast transit highways is apparent.

The *offset* styling parameter allows to displace a line symbol from its spatial delineation by a certain distance to the side (relative to line direction). This parameter had to be applied so that the symbols for bidirectional routes are both visible — otherwise the lines would overlap as the defining nodes of these segments have the same coordinates, it is only the node ordering that differentiates them. Styling across the zoom range has been applied to both the line width and the line offset to secure a reasonable graphic fill across scales (see Fig 3).

Fig. 3 The example of styling across the zoom range used in the application. The street line width is changing exponentially with scale (0.1px at zoom level 10, 2px at zoom level 14, 5px at zoom level 16). Screenshot from Mapbox Studio, a web-based tool to create and asses styles for vector tiles.

There is a range of cartographic methods that would allow comparison between the state of the traffic network in two moments. A dual map view or a difference layer would both be viable, though we wanted to employ some WebGl specific features of that would not be readily available in SVG or Canvas environments. For that matter, we chose to apply the *fill-extrusion* parameter combined with tilted camera view. Fill-extrusion is a method intended for use with polygon layers mainly to create 3D building models. But it is well applicable to line layers also. The *base height* fill-extrusion parameter allows to “lift” the shape off the ground, which enabled stacking multiple stripes on top of each other (Fig 4). Color coding of the fill-extrusion is still dynamically adjustable, which enables comparison of speed changes across weeks. This method certainly has its limitations: it is not well suited for global comparison, tilted camera view is necessary as well as user activity to pan and change the view angle. With bidirectional routes, each direction determines the color of one side of the 3D stripe, so the directions can not be viewed simultaneously.

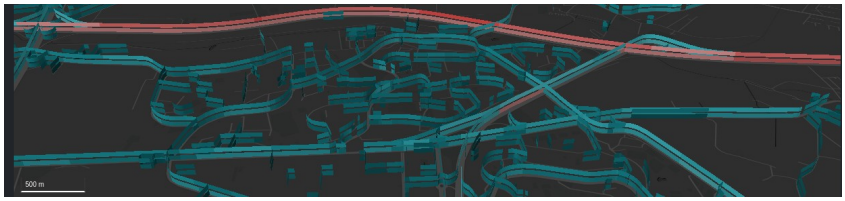

Fig. 4 Screenshot from the application in the comparison mode. The fill-extrusion parameter is used to support comparison across weeks.

5.4 User interface design

The resulting map based application works in two modes, the default *single-layer mode* supports displaying the traffic situation in a specified date and time. The *comparison mode* then allows to view differences between selected weeks. The main points of interaction in the default mode are the controls for selecting hour, day and week. There is a certain redundancy in those controls: user can either click the *back* and *forward* arrow buttons (right to the map in the large screen view), which allow to "jump" back and forth in time in fixed intervals. This enables observing speed changes throughout the day (by moving in the hourly interval), comparing the situation in given time across days (daily interval) or see the difference between weeks (weekly interval). The currently selected date, time and week are displayed in the numerical form on the right pane and are also visualised using the table below the map where rows represent weeks and columns represent days. The selected day is highlighted by the table field color. The table not only shows the temporal extent of the dataset and the current selection but also offers an alternative way to select the day and week by clicking the respective field. The slider below the map is coupled and aligned with the table. It provides a way to select an hour within the given week and also allows for smoother transition between map states.

In the comparison mode, which can be enabled by clicking the *compare* checkbox, some of the mentioned controls behave a bit differently. This mode allows for selecting two weeks which is facilitated by two drop-down menus. The hour and day selectors on the right pane remain active, whereas the week selector is disabled in favor of the drop-downs. The hour slider also remains active as it allows to change hour and day for both selected weeks simultaneously. It is also possible to change the observed day by clicking columns in the overview table (that has got two fields highlighted to denote the selected weeks).

A care has been given to ensure responsiveness of the interface layout. The map field, selection slider and table are sized dynamically by the screen width. On small screens the right-side control pane is moved below the map field to leave sufficient screen width for the map. The legend is fixed on the right side of the interface, which on the one hand places it out of the spotlight on large screens, but on the other hand it makes sure that the legend is placed next to the map field on small screen devices.

5.5 Evaluation and possible extensions

During the preparation phase, the tile size limit of 500 KB appeared as an unforeseen driving factor that influenced our decision making both in data and visualisation space. While there are alternative solutions like setting up a custom tile server, we chose to split vector tiles into chunks of the same spatial coverage but shorter time coverage to reduce the attribute count. This is a proven solution within the selected infrastructure, however more experimental approaches were tested over the course of the work. One of them is based on the fact that many road segments exhibited consequent runs of same speed values. The idea was to use a *run length encoding* algorithm to compress the attributes, which was

successfully done on the database side[14] — the encoded values were stored in an array type column in PostgreSQL and then exported as vector tile layers. However, decoding the values on the client side showed to be beyond the scope of the mapbox-gl style definition language. There is also a question of the rendering performance as decoding the run length array and finding the right value for the selected time would have to be done for each displayed segment.

Another solution would be to classify the streets in the dataset, mainly to separate the segments with low speed variability that could be represented by a single value for a longer then just the hour period. Overall, we identified three types of road segments in our problem area:

- Routes with low average speed and very low speed variability throughout the day. These are typically short segments with low traffic, cul-de-sacs leading to residential areas.
- Routes with medium speeds and visible daily variability. These are mainly the inner-city veins
- Routes with high average speeds and small daily speed variability. These are transit highways with high allowed speeds.

In this setting, it is hard to make any confident statements about the impact of the COVID-induced lockdown in the area. Only the second type of road segments seems to be susceptible to the impact of governmental restrictions. Also, the traffic speed is only an indirect indicator of traffic volumes. Lowered number of vehicles during the lockdown period could have contributed to higher speeds in otherwise notoriously congested areas on the main city lines and near highway entrance ramps. Another impact could be in

14 Script using *pandas* and *sqlalchemy* Python libraries available at https://github.com/pondrejk/dizzer/blob/master/misc/scripts/06-run_length_encode.py

evening out the daily changes in traffic speeds and lowering the significance of peak hours.

There are of course many areas in which the application could be extended. For example, summary data on individual road segments could be displayed upon selection. Additional modes of comparison could be added as well as a capability to search and filter the mapped data e.g. by the street name. A detailed overview of the speed changes across the user-selected could be implemented, possibly with comparison of multiple itineraries. Reader may surely think of other extensions. To conclude, the application underlines the potential of the vector tile format combined with WebGl based rendering environment to handle both spatially and temporally dense datasets. Depicting the subtle changes in traffic speed patters lets us at the minimum appreciate the collective organism or the city.

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)*.