Appendix 2d – Data preparation: Network blending

%Why network blending?

After all data was acquired, and all necessary data cleaning, filtering and conversion steps have been carried out, we need to bring the data together.

For the creation of the Detour Index (DI) and Local Significance (LS) indices, we need to produce one dataset that contains all network information and one dataset that contains the residential building and park entry points.

Additionally, these two datasets have to be harmonized to enable routing operations that are necessary for index creation.

We carry out these tasks during the network blending workflow.

%Network blending

To calculate the DI and LS indices, we require on the one hand information on the locations of the residential buildings and green space entry points.

One the other hand we need a network that connects the two.

In the previous data preparation steps, we have carried out all tasks to acquire and process these data.

Yet, we have three individual datasets that have no connections whatsoever.

The network structure that we use for our analysis consists of edges and nodes.

Each edge is a line that connects a start and an end node.

A node can have several edges emerging from it.

Two edges are connected, if they share the same node.

Each node will be given an index number in addition to its coordinates.

In turn, an edge will receive the index numbers of its start and end nodes.

Consequently, to facilitate network analysis with the data that we have acquired and prepared so far, we need to integrate the entry points of the residential buildings and the green spaces into the network.

Each line of the network data will be an edge in the network structure with nodes representing the starting and ending points.

Additionally, the building or green space entry points that have no relation to the network yet have to become nodes in the network structure, as well.

To integrate buildings and green space entry points into the network, for every point we have to i.) find the nearest point on the closest network edge, ii.) split the edge at that location and iii.) add a new node to the network.

The newly created node will receive all information from the original point and the index numbers of both, nodes and edges will be updated.

The creators of the R package *sfnetworks* utilize for this process the metaphor of throwing the data into a blender and mixing them together.

For blending points into a network, they have developed the *st\\_network\\_blend* function.

In our network blending workflow, we rely on the st\\_network\\_blend, as well as other functions from the sfnetworks package.

%Functionality

We have bundled the steps necessary to execute the network blending process in the *networkBlend* function.

The networkBlend function takes as input i.) the city code of the target city, ii.) an input directory containing the layer with the city boundaries, and iii.) the output directory of the previous data preparation steps containing the network data, the residential building entry points and the green space entry points.

The output directory will be used for storing the results of the network blending process as well.

The individual paths to the layers containing the output of the previous data preparation steps will be created automatically based on the output directory.

If desired, the paths can be set manually as well.

During the network blending process, we will follow the steps a.) dividing the data into grid cells, b.) snapping and blending the green space and residential building points data into the network, and finally c.) combining the grid cells back together:

\textit{a) Dividing the data into grid cells:} In the network blending process, we carry out a couple of computationally very intensive tasks. Especially the process of snapping a great number of points to a large network can use lots of resources. To increase the overall speed with which these tasks are being solved, we split the dataset into smaller chunks and distribute the work across multiple computation cores. These cores can now process the tasks in parallel, instead of having to perform them in series. By splitting the data, we not only save computation time by distributing the data, but we also greatly speed up the snapping process. Snapping means that a point is being ‘moved’ to the nearest point on the nearest line. If we reduce the size of the network, the snapping algorithm, instead of having to search the entire city network for the nearest edge, only has to search the reduced dataset. We parallelize the network blending process by overlaying the city boundary layer with a 2x2 km grid and keeping only those grid cells that intersect with the city boundary itself. The resulting grid cells, we pass on to the next function.

\textit{b) Snapping and blending the green space and residential building points into the network:} To run the computation for each grid cell from the previous function in parallel, we use the R packages *foreach* and *doParallel*. By default, the process will run on roughly 75% of all available computational core that the machine has available. Increasing this value could render the machine incapable of executing other tasks, which can lead to unexpected crashes, or, if run on a shared device (e.g. a server) spark social conflict. In the snapping and blending process, we convert the respective grid cell into a well-known-text filter (WKT). We now use the WKT filter to consecutively load the building and green space entry points and the network data that is located inside the grid cell into the machine’s memory. We load the network with a buffer of 100 meters, so any points that lie close to the grid cells edge have a change to be snapped to a street that is in the next grid cell. In a consecutive step, we merge the future nodes that should be blended into the network (i.e. the building and green space entry points) into a simple feature (sf) object. Now we hand the sf object containing all entry points and the network data that is inside the grid tile to the blending process. During the blending process, we utilize the sfnetworks R package. Particularly we use the as\\_sfnetwork and the st\\_network\\_blend functions. The former converts the network data that is located inside the respective tile into a sf-network. A sf-network is a combination of the sf data structure with the igraph data structure. In a sf-network, the edges and the nodes are stored with their respective data in two separate data frames but belong to the same object. The as\\_sfnetwork function converts the network data into a sf-network by converting each linestring geometry of the network into an edge of the sf-network and by adding nodes at every starting an ending point, as well as adding the required index numbers. The st\_network\_blend function now takes this network and the sf object with all the point geometries and blends them together. In short, the st\\_network\\_blend function entails:

i.)checking if a point geometry is located on an edge

ii.) if not: snapping the point to the nearest location on the nearest edge

iii.) and either: if the location is not already an existing node, splitting the edge and integrating the new node into the sf-network

iv.) or: attaching the points data to the existing node

A more elaborate and detailed description of the process can be found here: (\url{<https://luukvdmeer.github.io/sfnetworks/articles/sfn03_join_filter.html>#}). This method comes with a couple of drawbacks, though. If there are multiple points that share the same node location, only the first point will be used in the final sf-network. The same counts for a point having multiple possible locations on nearby edges. And lastly, the blending process can cause precision errors, where a node that is intended to represent the starting or ending point of an edge is a tiny distance away from the edges end point. This happens due to internal rounding errors in R and causes two connected edges not to be recognized as connected. To prevent this from happening, we have set the tolerance parameter of the st\\_network\\_blend function to 1 mm. After the blending process is complete, we write the resulting tiled sf-network into a temporary file, where it waits for further processing. As mentioned in the past paragraph, this process takes place in parallel for each tile of the city grid. So, we end up with several individual two files per grid cell. One file containing the edges and one containing the nodes.

\textit{c) Combining the grid cells back together:} After snapping and blending the points of the residential buildings and green space entries to the network, we end up with several files containing edges and nodes. For the network analysis, one file containing all edges and one containing all building and green space entry nodes of the city. Combining the nodes is rather simple. We consecutively read all individual files containing the nodes and make sure all geometries are correct. Afterwards, we assign all columns the correct data type to make sure they can be merged together and append all files into the same geopackage database. The combination process of the edge files has to consider that edges can cross borders of grid cells, so we are likely to have created duplicated edges close to the borders. To remove these duplicates, we utilize the st\\_network\\_join function of the sfnetworks R package. The st\\_network\\_join function performs a join of the network data, making sure duplicates are removed to a minimum. Nonetheless, we utilize the remove\\_overlap function from the network cleaning data preparation step to further reduce the number of overlaps. Using redundant functions in this step has proven to enhance the quality of the network that we later use for the network analysis. In a final step, we also write the combined edges to their geopackage database and delete the tiled edge and node data to reduce disk usage.

At this point data preparation is finished. The required data that we need to create the indices is cleaned and in the right format to serve as input into the network analysis.