Appendix 2a – Data preparation: Network cleaning

Why network cleaning?

* OpenStreetMap (OSM) is a large and heterogeneous dataset that is being audited by the OSM community.
* Unfortunately, the immense amount of data and heterogeneity of the OSM contributors are a guarantee that the data is not ready to use.
* In cleaning the network prior to taking further steps in the analysis, we ensure a higher consistency of the results.

Network cleaning

* A variety of errors can cause problems when using network data for an analysis.
* One common error in network data is small deviations in the coordinates of lines that are supposed to be connected.
* It often occurs that the coordinates that make up the endpoint of the first line and the starting point of the next line are *almost* the same.
* If the point coordinates that make up the lines are stored with too much precision, a small deviation between these two points, can cause the lines to be found unconnected by our analysis tools (plot).
* Other common mistakes are duplicated nodes or crossing lines that have no common node at the intersection and are thus not connected.
* On top of the usual pitfalls of working with network data, we have downloaded city-wide data in tiles of 2×2 km.
* Prior to the analysis, these tiles have to be joined together to create a network covering the entire city.
* On the edges of these tiles, there may be overlapping paths due to e.g. a street crossing through the border of one of our tiles.
* This street would be downloaded twice in the two adjacent tiles.
* For a consistent analysis, we have to make sure that most of these errors are being detected and corrected.
* This task is being carried out by the *networkPrep* function of our script.

Functionality

* The cleaning process is made up of the *networkPrep* function, which itself relies on a plethora of different sub-functions.
* The networkPrep function takes as input i.) the city code in URAU format, ii.) an input directory containing the required layer with the city boundaries and the network tiles that were downloaded by the download\_OSM function and iii.) an output directory for saving the output.
* The directories containing the required input can be set manually or will be guessed automatically from the in- and output directories.
* The network cleaning process follows these steps: spatial and thematic filtering, combining the network tiles and cleaning the network

1. Spatial and thematic filtering
   1. The first step of the network cleaning process involves loading the city boundary that is corresponding to the URAU code that was provided to the networkPrep function.
   2. Since we downloaded the OSM data in rectangular boundary boxes, we downloaded a ‘surplus’ of data at the tiles overlapping the edges of the city.
   3. The city boundary will later be used to spatially filter out all information that is located outside of the city.
   4. Thus, the amount of data that is stored on the machine will be reduced.
   5. On loading the city boundary, the networkPrep function will also apply a buffer of 1000 meters to the boundary.
   6. The buffer ensures that the cleaned network will later cover areas outside the cities border that households living at this border can visit inside a 500 meters walking distance.
   7. In the next step, the networkPrep function will list all tiles that have been downloaded and whose names match the cities URAU code.
   8. During this process, the boundary file will be transformed into a well-known-text (WKT) filter.
   9. The WKT filter will be used to load only those parts of the downloaded OSM data from file that are needed because they are located inside the city boundaries.
   10. In addition to spatially filtering the OSM data, the script will automatically discern the *highway* column in the OSM data from the many other columns that contain the string ‘highway’.
   11. The *highway* column contains the information describing the importance of the path in the network (e.g. motorway, primary street, residential road etc.).
   12. All path with the value ‘motorway’ in the highway column will be filtered out (https://wiki.openstreetmap.org/wiki/Key:highway).
   13. Motorway was the only value that we could consistently identify as the only class of network elements to be accessible only for cars and not for pedestrians.
   14. For example, filtering out primary roads, even though they are supposed to ‘often link larger towns’, led to missing important walking axis that are accessible to pedestrians in large German cities (Mehringdamm plot).
2. Combining network tiles
   1. Now the actual network combination process starts.
   2. After spatially and thematically filtering the network data, it will be casted to linestring geometries.
   3. This way we make sure that no other geometry types will cause errors in the further analysis.
   4. The pre-cleaned network tiles will now be written into the same layer of a Geopackage file.
   5. Appending the individual tiles to the same layer of one file has proven the most efficient way of combining the network tiles.
   6. Other ways of combination would include loading all network data into the RAM of the machine and combining it there, taking a larger amount of time.
   7. After all files of downloaded OSM network data are written to disk, the network data is ready for further cleaning steps.
3. Cleaning the network
   1. In the last part of the network cleaning workflow, the actual pitfalls of network data are being tackled.
   2. The network cleaning script heavily relies on the tidygraph and the sfnetworks R-packages and loosely follows the sfnetworks pre-processing and cleaning tutorial by Lucas van der Meer, Lorena Abad, Andrea Gilardi and Robin Lovelace (<https://luukvdmeer.github.io/sfnetworks/articles/sfn02_preprocess_clean.html>).
   3. Further cleaning of the network dataset involves i.) removing double entries, ii.) rounding coordinates, iii.) subdividing edges, iv.) spatial smoothing, v.) removing unconnected edges and vi.) removing overlapping edges.
      1. Removing double entries: In the first cleaning step, all double entries in the now combined network dataset are being removed. This process uses the dplyr ‘*distinct’* function which only removes double entries from the data frame without touching the geometries of the data. In this step there will not be removed any overlapping edges, yet.
      2. Rounding coordinates: Now we make sure all coordinates that make up the lines of the network data are exactly the same where lines are supposed to be connected. For this correction, we chose to reduce the precision of the network data. We did so by applying a rounding function to the coordinates of the line geometries. This process will implement a minor imprecision of a few centimeters in the network data. And too much rounding can cause lines of zero length which will cause errors in later analyses.
      3. Subdividing edges: In the later analysis, we rely on the sfnetworks package. When creating a network from OSM network data with this package, the linestrings of which the network consists will make up the edges. The endpoints of these edges become the nodes of the network. Edges that share the same endpoint also share the same node – i.e. they are connected. An edge can still have interior points that define its shape but are not nodes of the network. If an edge shares exactly the same interior point with another edge but this point is not a node, they are not being considered connected in the network structure. To rule out any unconnected edges due to this error, we can split the overlapping edges at their common point. This point, in turn, becomes a node of the network and the edges are connected. The function *to\_spatial\_subdivision* of the sfnetworks package in combination with the tidygraph function *convert* does exactly this. All edges are be connected at shared interior nodes. Edges will not be connected if they are overlapping but do not have any shared nodes, reducing the danger of connecting bridges to the paths that are lying beneath them.
      4. Spatial smoothing: A spatial network might as well contain nodes that do not play any connecting role in the network structure. These nodes have only one incoming and one outgoing edge. These pseudo nodes would only increase the complexity of further analysis and, thus, increase the time it takes to compute the results. To remove pseudo nodes, the sfnetworks package offers the function *to\_spatial\_smooth*, which can be used with the tidygraph function *convert*. After removing a pseudo node, to\_spatial\_smooth will transform the two adjacent linestring geometries into a single edge.
      5. Removing unconnected edges: Another common pitfall in the downloaded OSM network data are unconnected edges that form their own network structure. Edges can be unconnected from the rest of the network due to several reasons. They can, for example belong to paths that are located inside of a courtyard or atrium that has no publicly available connection the rest of the network. Leaving unconnected edges in the network dataset would in the later analysis cause unreachable areas of the city. E.g. residential building entries could be assumed to be located in these areas and thus vanish from the analysis. To counteract this source of errors, we filter the network data using the tidygraph *group\_components* function, which groups a network by its connected components. We use this function to keep only those edges that are connected to the largest component – i.e. the city network.
      6. Removing overlapping edges: In the last step, we aim to remove any edges that are still overlapping from the network. An overlap could, for example, be an artifact of downloading the OSM network data in 2x2 km tiles. If a street is crossing the boundary between two of the tiles, part of it might be downloaded once for each tile. Downloading the street twice will result in two longer edges with ending parts that are overlapping (plot). Overlapping edges would cause the same path to appear multiple times in the network. This could potentially cause the local significance index values, which are supposed to accumulate on the edges, to distribute across several overlapping edges. For removing overlapping edges, we have devised the *remove\_overlap* function. Remove\_overlap relies on the R- package *sf* and its *st\_covers* function (sf package). To remove overlapping edges, remove\_overlap will separate the network into those edges that cover multiple other edges and edges that only cover themselves. Consecutively, it will split the edges that are covering other edges and only keep those parts that form a unique spatial feature.