Appendix 2c – Data preparation: Green space entry detection

Why green space entry detection?

* To measure the distance from a building entry to the nearest green space, we need a fixed point that demarcates the nearest entry point to that green space.
* Unfortunately, the data that we have acquired does not provide reliable information on those entry points.
* Consequently, we have to estimate where the entry points of each park in a city might be located on a network.

Green space entry detection

* To detect green space entry points as reliable as possible, we chose to intersect the urban atlas (UA) polygons of the classes urban green (14100) and forest (31000) with the OpenStreetMap (OSM) network data.
* We used the outline of the UA polygons so the intersection algorithm only yields points that are on the edge between green space and city.
* During the data preparation workflow, the *greenSpacePrep* function in R will take on this task.
* When detecting green space entry points, the greenSpacePrep function will apply increasing buffer sizes until each green space in a city has at least one entry point.
* Furthermore, we have to account for inhabitants that live at the edges of poly-nuclear cities and are thus able to travel to green spaces across city borders.
* To avoid this problem, we have to integrate the UA green spaces of adjacent cities into the analysis.

Validation

To refine the approach of green space entry detection, we used the outlines of green space polygons with buffers of different sizes applied to them.

* We selected three Berlin green spaces to validate the accuracy of the different buffer sizes.
* To cover a large variety of urban green spaces with the available work force, we chose the Tempelhofer Feld, the Viktoriapark and the Treptower Park.
* In the Viktoriapark, park entries are well defined by surrounding walls / greenery.
* The Treptower Park is rather open and has lots of meadows at its edges.
* Furthermore, the Treptower Park is delimited by the river Spree on its northern side and a train station at its western end.
* The former airport Tempelhofer Feld is in the original UA data set classified as ‘sports and leisure facilities’ but fulfills in our opinion the function of an urban green space.
* The Tempelhofer Feld is delimited entirely by a fence.
* Entry is only possible at gates that are designated for public entry.
* At each of the three sites, we recorded the GPS coordinates of each entry point.
* We tried to mark the locations where people are expected to enter the green spaces, e.g. paths leading into a park, gates, bridges or station exits.
* We used the resulting GPS points to estimate how much over- or underestimation was produced by automatically detecting green space entry points with the different buffer sizes.
* To account for the different buffer sizes, we manually selected edges from the OSM network data demarcating possible entry trajectories into the green space for each GPS point.
* We used Google Earth Pro aerial photographs and QGIS to mark the network paths serving as entrance trajectories.
* The different buffer sizes around the green space polygons we tested are: -10, -5, 0, 5 and 10 meters.
* For the following sensitivity analysis of the different buffer sizes, we checked for each validation trajectory whether or not it was detected with the utilized buffer size.
* ‘Detection’ in this case means that the ‘detected’ park entry was inside a buffer of 1 meter around the validation trajectory.
* We measured the accuracy of a buffer size by assessing if green space entries were correctly detected (validation data) without producing pseudo entries (e.g. at intersections with the street network that are no entry points).
* We calculated the highest producer accuracy for the buffer sizes of negative 5 meters and 0 meters.
* This means that the algorithm was equally good at detecting true positive green space entries with both buffer sizes.
* The commission error was lowest for negative 5 meters and increased in both directions.
* So when applying a lower of higher buffer, the algorithm falsely detected green space entries that were not in the validation data.
* High producer accuracy and low commission error resulted in the highest overall accuracy for a buffer size of a negative 5 m around the green space polygons.
* The second most accurate buffer size was a buffer of 0 meters.
* With increasing buffer size, the accuracy of the detected park entries decreased and the overestimation of park entries increased further.
* A possible explanation for this outcome might be that most of the green space polygons align with the nearest larger streets.
* A negative buffer will in this case better detect the paths that are emerging from these streets.
* Of cause, these results have to be taken with a grain of salt, since we only validated them in Berlin and only with a handful of green spaces.
* There is by far no guarantee, that these findings can be readily translated to other cities or countries.
* When looking into the outcome, this should always be considered.

Functionality

* In the data preparation workflow that we wrote in R, the greenSpacePrep function will assign at least one entry point to each green space of a city.
* The greenSpacePrep function takes as input i.) the city code in URAU format of the respective city, ii.) the input directory that contains the city boundaries layer and the UA and network data, and iii.) an output directory for storing the resulting green space entry points.
* The paths to the required input files will be guessed automatically but can be provided manually, as well.
* The greenSpacePrep function follows the steps a.) checking the proximity to other cities, b.) loading the UA green space from all cities in question, c.) detecting green space entry points and d.) rounding the geometries and outputting the data:
  1. Checking proximity to other cities: When checking the proximity to other cities, we start by loading the target city boundary from the city boundaries layer that corresponds to the URAU that we used as input. After loading, we will apply a buffer of 1000 meters to the city boundary. This way we make sure that any green spaces that are inside a network distance of 500 meters from any residential buildings in the city will be factored in later. Now we will check if there are any other cities from the layer with all city boundaries intersecting with the buffered city boundary. We do this again by first converting the buffered city boundary to a well-known-text filter (WKT). The WKT filter is then used to extract the functional urban area (FUA) code from any other city that intersects with the target city boundary. We use the FUA code here, because the UA file names are encoded in this style. Several URAU coded cities can share a single FUA code. If any neighboring cities were detected, the respective FUA codes will be passed on the next function. Otherwise, we will proceed with only the URAU code of the target city.
  2. Loading UA green spaces: When loading the UA green space data, we initially filter the directory with all UA city data for the directories matching the city codes (URAU and / or FUA) that were passed on from the previous function. Upon iterating through the respective UA data directories, we use a WKT filter representing the buffered target city boundary to load only the UA data into the machines memory that is located inside the target city. To further reduce the data size, we select the columns containing the FUA code, the UA class code, the UA identifier and the area, i.e. the size of the polygon. We filter the resulting data to obtain polygons identified with the UA classes 14100 or 31000. If a future user desires to use UA data from 2006, be wary of the fact that the forest code in 2006 was 30000. Furthermore, any further areas that are desired to be taken into account in the following analysis can be added here. For example, we have manually added the Tempelhofer Feld in Berlin, because it is classified as ‘sports and leisure facility’ in UA. We consider the Tempelhofer Feld to fulfill the same role as an urban green space, though. The resulting green spaces are gathered in a temporary file and passed on to the next function. On request, an output directory can be provided.
  3. Detecting green space entry points: When detecting the green space entry points in a city, we first load the respective network data that we have cleaned during the network cleaning process. Now we apply the negative 5 meter buffer to the UA green spaces that we obtained in the previous step. To prevent any geometry errors during the further analysis, it is advisable to make sure all green space polygons are of the correct geometry type (Polygon). In the next step we cast the negatively buffered green space polygons to linestring geometries. This way we retrieve the outlines of the green spaces which we can now intersect with the network data. Some green spaces might be to far from the next network edge or by chance their outlines did not intersect with the network. To make sure all green spaces receive at least one entry point, we check if there are any green spaces left without. We iteratively increase the buffer size in steps of 5 meters, intersect the remaining outlines with the network and check again. To reduce computation time, we increase the steps to 10 meter for a buffer size between 50 and 100 meters, 25 meters for buffer sizes between 100 and 200 meters and 50 meters for buffer sizes larger than 200 meters. When no green space is left without an entry point, we pass the result on to the next step.
  4. Rounding geometries: Before writing the green space entries that we detected in the previous process to file, we round their geometries. This has proven to prevent R from randomly crashing during the further analysis. If not provided by the user, the greenSpacePrep will automatically generate an output file name and store the green space entries here for further use.