ADVANCED DATABASE SYSTEMS

2019 PROJECT TOPICS

RULES AND GUIDELINES

1. Projects will be carried out by groups consisting of 3–4 persons.
2. Each group will get access to a GitLab project, which should be used for project management (issues/board), documentation (wiki) and source code management (git repository).
3. Unless specified otherwise or required by the specificity of a particular topic, the implementation should be minimalistic, i.e. require minimum environment setup and be easy to launch and evaluate. The preferred form is a Jupyter Notebook or a similar solution. Other forms must be agreed upon with the supervisor of the given topic.
4. The projects should support evaluation of results, e.g. using a Leaflet widget inside Jupyter (for map-related topics).
5. The implementation should run on Linux and macOS and include setup instructions. In case of Jupyter Notebooks, this is only required for packages not available in PIP (or those which require additional setup steps).
6. The code should be provided in the master branch of the repository.
7. The documentation should include instructions, a description of the assumptions and algorithms used and a statement regarding the contributions of individual team members.

PROJECT TOPICS

TOPIC 1: WMS DATA ANALYSIS

WMS services provide map data as bitmaps containing data from selected layers. They also allow for querying a given location for objects' parameters, but they can't provide a list of objects existing within a given region. The goal of this project is to create a piece of software which:

1. downloads WMS tiles for a given layer and region,
2. checks if objects exist in a given location,
3. queries the service for the objects' parameters,
4. returns the obtained data as a PostGIS/SpatiaLite table.

Attention must be given to the following aspects:

- determination of optimal map scale,
- choice and understanding of the CRS used,
- scheduling of queries so as not to overload the WMS service,
- parsing of the parameter data (which can e.g. be an HTML table).

TOPIC 2: OSM VS. OTHER MAPS

Your job will be to match OSM data for a given region (e.g. city) with selected layers of other geodetic data, such as the Polish BDOT (Topographical Object Database) dataset. For example, in case of streets, each way representing w street in OSM should be matched to an object in the other database. Please note that there can be caveats:

- One object in OSM may correspond to multiple objects in the other set, or vice versa.
- The shape and location of objects may vary slightly.
- Some objects may exist in one dataset but not in the other.

The result should present the matched object pairs and the difference of corresponding parameters (e.g. road width), as well as those objects which could not be matched.

TOPIC 3: WHAT IS LIT?

You will receive a dataset with lamp locations – each lamp post represented by a single point (for example: https://gist.github.com/e4d74603905701a908ec9f32613e6520).

Download OSM data for the corresponding region. Your goal will be to identify the OSM ways representing roads which are lit, e.g. which have lamps nearby. Note that some ways may only be partly lit – in such case, you should provide the lit range of the way (e.g. 0–0.67 lit, 0.68–1 not lit). For lit roads or road fragments, calculate the average lamp spacing in meters and the total number of lamps. For all road fragments, calculate the length of the fragment. Compare the obtained results with the *lit* OSM attribute values for these roads.

TOPIC 4: OSM SIMPLIFICATION

Your job will be to generate a simplified version of a given OSM dataset by reducing the number of ways though merging. The reduction should be carried out in the following steps:

1. Determine the subset of ways included, e.g. only those with the *highway* attribute, and filter the ways.
2. Determine the attributes which should differentiate the ways (i.e. prevent merging), e.g.: name, highway type, lane count.
3. Adjacent ways with identical values of attributes specified in the preceding step should be merged into one, preserving other possible attributes which are common for all merged ways.

Caveats:

- two mergeable ways may have different directions, which means some may need to be reversed to form one new sequence; direction-specific attributes (e.g. *oneway*) should be adjusted accordingly,
- in some points, more than two mergeable ways may “meet”; obviously, merging needs to take place in pairs.

The input and the output for the project are OSM (XML or PBF) files. The recommended (but not required) approach is to load the input file into a PostGIS database using Osmosis, perform the matching and dump the result (via Osmosis) to the output file.

TOPIC 5: POSTGIS GEOJSON INTEROPERABILITY

PostGIS allows for serialisation and deserialisation of geometry objects to GeoJSON, but that's limited to simple structures without properties. Your task is to create a solution which will allow:

1. exporting any PostGIS table to a GeoJSON file, where the geometry is taken from a specified column and all other columns are saved as properties,
2. creating a PostGIS table based on an existing GeoJSON file.

The preferred approach is to use database-side procedures. Since GeoJSON files can have multiple levels of hierarchy, the user should be able to choose the granularity and aggregation level when importing GeoJSON files. The solution should "intelligently" guess the appropriate column type based on the values of JSON properties.

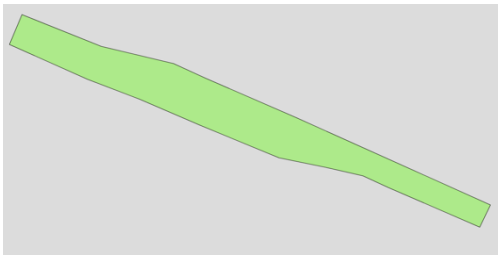
TOPIC 6: SHAPE APPROXIMATOR

Consider two types of geospatial shapes representing roads:

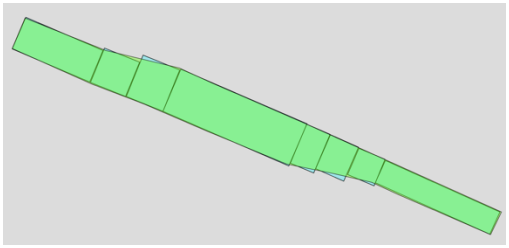
1. line-based shapes – polygons created by buffering linestrings with some radius value (https://postgis.net/docs/ST_Buffer.html),
2. freeform shapes – any polygons, possibly providing a more accurate representation of the road shape.

The goal of the project is, having a freeform shape, to find its closest approximation using a set of rectangle shapes.

Freeform shape (left):



Freeform shape approximated with a set of rectangle shapes (desired result):



Bulk download example freeform shapes OpenStreetMap. Use objects which have the *area:highway* tag. You can use the Overpass Turbo API (https://overpass-turbo.eu) to locate the appropriate shapes. The tests should involve at least 100 different shapes. Some examples:

- https://www.openstreetmap.org/way/467739486
- https://www.openstreetmap.org/way/511438520
- https://www.openstreetmap.org/way/511346012

TOPIC 7: CRAWLER

Create script that, for a given website, generates a graph in Neo4j of all (or a reasonable amount of) subpages and their connections. Each node should contain the URI, title, summary of content (extracted from h* tags, title, etc.).

TOPIC 8: BENCHMARK

Benchmark comparison of databases of 3 types - PostgreSQL, CouchDB (or other NoSQL), Neo4j.

1. storing graph representation – speed of queries, searching paths, loading data,
2. storing geospatial data – basic operations – calculating distances, areas, speed of queries.

TOPIC 9: OSM TO NEO4J

Create a tool that allows for importing data from OSM files into a Neo4j database. You need to design the structure of the graph that will hold OSM objects, including ways, nodes and relations.

Compare your solution with existing solutions.

TOPIC 10: ROAD AND PAVEMENT SHAPES

The goal of this project is to generate shapes (polygons) representing OSM streets. For that purpose, you will receive

- a dataset containing most road edge shapes
  (for example: https://gist.github.com/bc3be021cda21b8ea1cc5350bf86a6e7),
- a dataset with pavement edge (curb) shapes
  (for example: https://gist.github.com/c57dca06c17ca05346872bd0b8abaeba).

Your job is to download OSM data for the corresponding region and, for each street (way):

- estimate the left and right edges using the provided datasets,
- add the front and back edges as straight lines, thus generating a polygon representing that street,
- check for pavement boundaries parallel to the street edges and, if found, generate polygons for that street's pavements as well.

TOPIC 11: CITY STATISTICS

The goal of the project is to download and analyse OSM data for a given administrative region (e.g. a city, a district, etc.).

As the input, you will get the OSM ID of the relation representing the region (e.g. 2768922 for Kraków). Then, your script should:

1. Estimate the bounding box which contains the region's boundary using the Overpass Turbo API, adding a margin of a certain width.
2. Download the appropriate smallest OSM dump from GeoFabrik.
3. Crop the dump accordingly and import into a PostGIS database (both using Osmosis).
4. Calculate the length of roads which fit within the boundary, grouped by OSM highway type.
5. Present the statistics as a data structure or Pandas dataframe.