ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

France National School of Geographic Sciences

Finnish Geospatial Research Institute, National Land Survey of Finland

Funding bodies

Internship report

ENSG Engineering Cycle — 2nd year

Massive data contributions from the National Land Survey of Finland to the OpenStreetMap Database

Alexys Ren

May - August 2023

☒ Non confidentiel ☐ Confidentiel IGN ☐ Confidentiel Industrie ☐ Jusqu'au ...

Ecole Nationale des Sciences Géographiques
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Jury

Sponsor:

Prof Juha Oksanen, director, Geoinformatics and cartography, NLS/FGI

Internship supervision:

Juha Oksanen (and other department's senior scientists)

Referring teacher:

Victor Coindet, IGN/ENSG/TSI

Head of engineering studies:

Jean-François Hangouët, IGN/ENSG/DIAS

Internship management:

Isabelle Joyeux, IGN/ENSG/DSHEI

Second year Multidisciplinary Internship from 22/05/2023 to 11/08/2023

Webcasting: ☒ Internet ☒ Intranet Polytechnicum ☒ ENSG Intranet

Document status:

End-of-year internship report presented at the end of the 2nd year of the engineering cycle

Number of pages: 92 pages including 50 pages of appendices

Host system: LaTeX

Modifications:

EDIT	OVERHAUL	DATE	MODIFIED PAGES
1	0	06/2023	Created

Acknowledgements

First of all, I would like to thank Juha Oksanen, my internship supervisor. He, together with Valtteri Nikkinen, Project Coordinator, gave me a very warm welcome and made sure that I had everything I needed to feel comfortable. Thanks to them, I was able to quickly adapt to this new environment and try to make the best of it.

I'm very grateful to Ville Mäkinen, Research Group Manager, and Jaakko Kähkönen, Senior Research Scientist, who, along with Juha, were very helpful in untangling technical problems during my research and were themselves available whenever I needed support.

I would also like to thank Ms. Ana-Maria Olteanu-Raimond, Senior Researcher and Co-Director of the French Lastig Laboratory, who introduced me to Juha and provided me with the basic material I worked on, without which this internship wouldn't be possible.

I'm grateful to Victor Coindet, my referring teacher at ENSG, for his helpful feedback and suggestions that helped me make the best of my work.

I'd also like to thank the Fondation ENSG-Géomatique, Coexya SAS and the European Union for their financial support of this project, which allowed me to carry out my research under the best conditions.

And finally, I would like to reiterate my thanks to all of FGI. I've spent such a great moment in this atmosphere, every person I've met has been wonderful. Thank you — Panu, Pyry, Valtteri, Anssi, Ville, Camilo, Milla, Justus, Hilla, Alpo, Anna, Rebecca, Jaakko, Christian, Juha.

Résumé

(French Abstract, English version next page)

Le FGI, ou l'Institut finlandais de recherche géospatiale, est situé dans la ville d'Espoo, à l'ouest de la capitale Helsinki. Il mène des recherches novatrices et des travaux d'experts dans le domaine des données spatiales et, plus précisément, traite des services web interopérables, des normes techniques et de l'harmonisation des données spatiales, qui sont les principaux domaines du département de géoinformatique et de cartographie dans lequel je suis formé.

Dans le prolongement de l'harmonisation des données spatiales, qui vise à standardiser l'information géographique au sein de l'Union européenne, l'objectif de mon travail est d'identifier les contributions massives du NLS (organisme supérieur dont relève le FGI) à la base de données OpenStreetMap afin d'obtenir un état des lieux des données NLS sur Internet, ainsi qu'une évaluation préliminaire des risques, ceci dans l'espoir de contribuer à de futures études sur le potentiel des entités importées ayant subi des modifications géométriques pour améliorer la base de données topographiques du NLS.

Notre étude commence par l'obtention des données d'intérêt auprès de fournisseurs de données bien connus tels que Geofabrik.de et Planet OSM, avant de présenter quelques techniques de manipulation des données OSM à l'aide de Osmium Tool, le pilote OSM de la librairie GDAL, Python et SQL (PostgreSQL), en préparation de la phase d'analyse des données.

Les premiers résultats permettent de supposer l'emplacement des bâtiments liés au NLS ayant des contributions massives dans la région finlandaise d'Uusimaa, ainsi que l'emplacement de ceux qui ont subi des modifications géométriques.

Mots clés : OpenStreetMap, importation massive d'OSM, analyse des données d'OSM, clé source d'OSM, import du fichier historique d'OSM, ouverture des données du NLS Finlande au grand public

Abstract

The Finnish Geospatial Research Institute (FGI) is located in the city of Espoo, west of the capital city of Helsinki. It conducts innovative research and expert work in the field of spatial data, and more specifically, deals with — interoperable web services, technical standards and harmonization of spatial data — which are the core areas of the Geoinformatics and Cartography department in which I train.

Following on from the harmonization of spatial data, which aims to standardize geographic information across the European Union, the goal of my work is to identify the massive contributions from the NLS (higher body under which the FGI is submitted) to the OpenStreetMap database to get a situation picture of the NLS data on the Internet, as well as part of a preliminary risk assessment, this in the hope of contributing to future studies on the potential of imported features that have undergone geometric modifications to improve the NLS topographic database.

Our study begins by obtaining the data of interest from well-known data providers such as Geofabrik.de and Planet OSM, before presenting some OSM data manipulation techniques using — Osmium Tool, GDAL OSM Driver, Python and SQL (PostgreSQL) — in preparation for the data analysis phase.

First results help to assume the location of NLS-related buildings with massive contributions in the Finnish Uusimaa region, as well as the location of those that have undergone geometric modifications.

Key words: OpenStreetMap, OSM massive import, OSM data analysis, OSM source tag, OSM history file import, NLS of Finland open data release

Contents

Glossary and useful acronyms

Apothem Distance from the center of a regular polygon to one of its sides

Capital Region Consists of four municipalities — Helsinki, Vantaa, Espoo and Kauniainen

Contributor Refers to any user who contributes to the OpenStreetMap community by importing its data (personal data or from external sources)

ENSG *École Nationale des Sciences Géographiques* — FRENCH. Standing for France National School of Geographic Sciences

Feature Synonym for an OSM element

FGI *Finnish Geospatial Research Institute*. Conducts innovative research and expert work within the field of spatial data. Under direct supervision of the NLS and place of the internship (Geoinformatics and Cartography Department)

Helsinki Metropolitan Area Not firmly established term, varying in different contexts. Commonly refers to the Greater Helsinki, the metropolitan area surrounding the capital city of Helsinki, and including the smaller Capital Region

IGN *Institut national de l'information géographique et forestière* — FRENCH. Public administrative institution under the supervision of the French Ministries of Ecology and Forestry. Its mission is to produce and distribute reference data (open data) and representations (online and paper maps, geovisualisation) relating to knowledge of the national territory and French forests, as well as their evolution

NLS *National Land Survey of Finland*. Official body dealing with cartographic and cadastral matters in Finland under the direction of the Finnish Ministry of Agriculture and Forestry. Equivalent to the French IGN

OSM *OpenStreetMap*. Collaborative online mapping project that aims to build a free geographic database of the world, using GPS and other open data. Launched in July 2004 by Steve Coast at University College London

Introduction

In 2012, the National Land Survey (NLS) of Finland opened its topographic data to everyone, thus allowing a large amount of data downloading from all over the world. Yet, with the ongoing war in Ukraine, there has been a change in the way of thinking regarding the transparency of map data. Already last year, there were a lot of requests from Russia for municipal map services, in line with its strategy of using open source data in the offensive. This prompted some politicians to call for more restricted spatial data accesses, especially those about the locations of critical infrastructures [17].

Even though there is no sudden threat that could be directed at Finland, it is justified to take a closer look at what information is made available and how it is used. Our case study will focus on OpenStreetMap, one of the biggest open spatial database in the world, where interesting information is most likely to appear. As part of a preliminary risk assessment for NLS data and with this in mind, we will try to answer the following question: **What massive contributions from the NLS has been done to OSM in history?**

More specifically, the goal of my work is *to discover how the NLS's data has been used in contributing to OSM database since the NLS's open data release*. We will investigate what was the data sent to OpenStreetMap from the NLS's perspective and what modifications those data underwent.

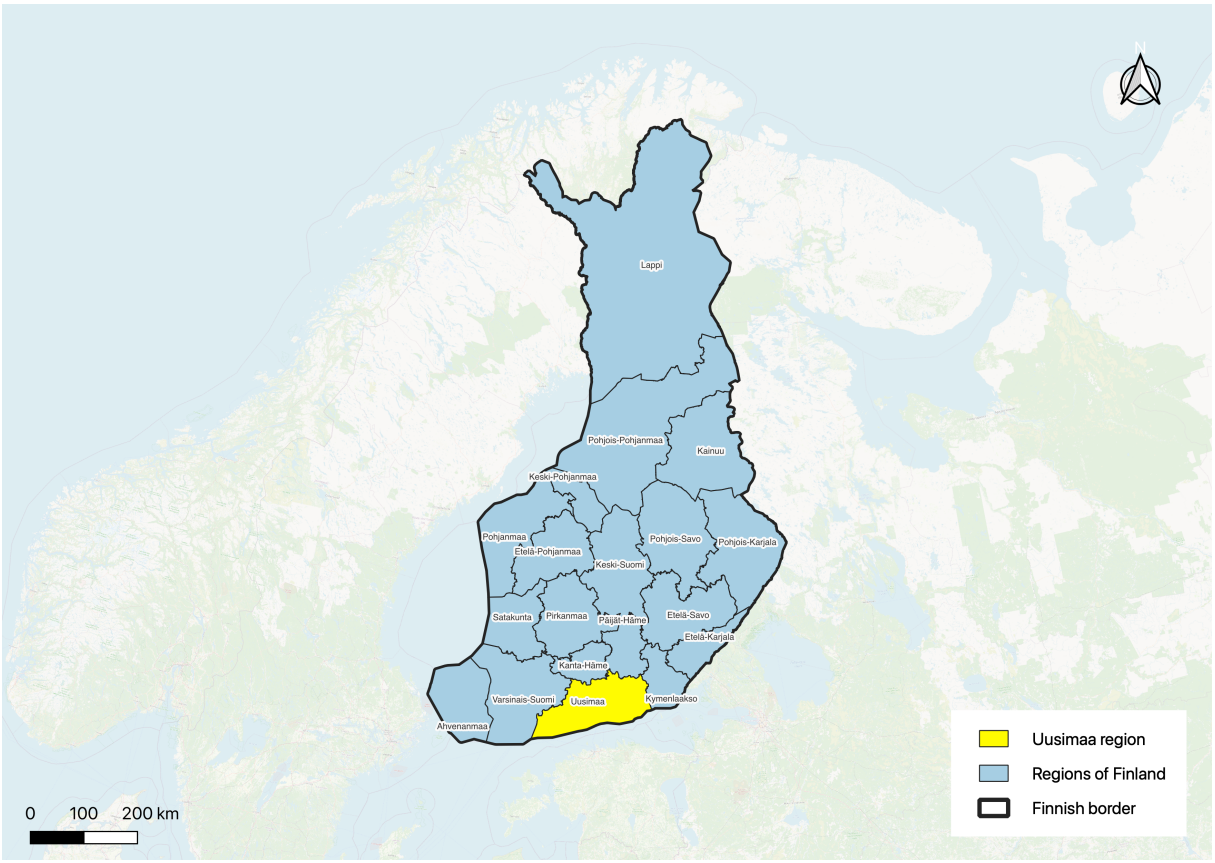
In addition to being an important investigation of getting a situation picture of the NLS's data in OSM, this work is also an opportunity for Prof. Oksanen to strengthen the collaboration between the French IGN and the Finnish NLS FGI across exchanges with Ms. Ana-Maria Olteanu-Raimond (Senior Researcher and Co-Director of the French Lastig Laboratory), with who they both planned to collaborate on summer 2022. On the other hand, NLS will be glad to have a better understanding on this matter, as this will allow the organization *to get understanding on OSM's potential to improve NLS's data with its latest updates*.

To this extent, my whole work structure will be based on Le Guilcher's, Olteanu-Raimond's and Bailo Balde's paper published in 2022 [19]. *The idea is to try to follow as carefully as possible the method her team discovered by applying it to Finland*, in order to enable the possibility for making comparison with the results obtained in France. We'll look on introducing the study area and the data choices first, before retrieving data from OSM. Then, we'll proceed with cleaning up the data and the data handling techniques, and eventually with both analyzing the data and getting visualizations of NLS features.

Chapter 1
Study site and data sets selection

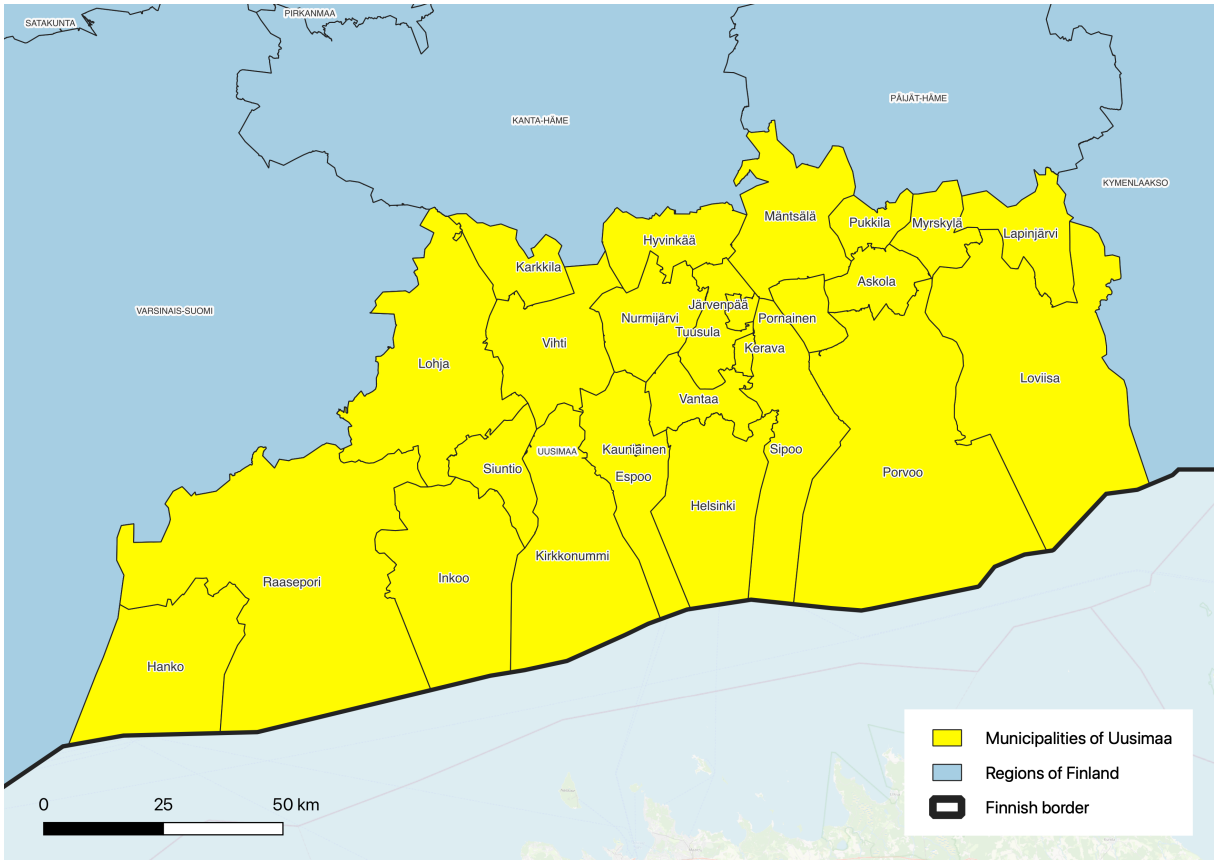
1.1 Study site

On Prof. Oksanen's suggestion, the study area will be limited to the *Uusimaa region and its surroundings*. The extent of the area is about 10,000 km^2 wide and includes the Helsinki Metropolitan area, which should be sufficient for preliminary tests.



(a) Location of the Uusimaa region in Finland

Figure 1.1: Finland's Uusimaa region (Map base credits: © OpenStreetMap contributors)



(b) Focus on the Uusimaa region

Figure 1.1: Finland's Uusimaa region (cont.)

For the sake of brevity, the study site will be referred to as *Uusimaa* for the rest of our reasoning.

1.2 OSM data structure

The OpenStreetMap data is made of core components which are the *elements* [5]. These features correspond to specific geometries which are among one of those three following types:

- node: points defined by their coordinates
- way: polylines defined as sequences of 'nodes'
- relation: features grouping one or more 'node', 'way' or 'relation'

Each element is also characterized by a list of tags (pairs of key/value) giving more information about it. Some elements may have mandatory tags (for instance the feature ID).

1.3 Identifying NLS data among the whole OSM data

Searching for NLS-related data is done via the *source* tag [9], as described in the first step of the methodology in [19]. It is used to indicate the source of some piece of information in OSM, and can help contributors who wish to understand the origins of the data they are editing.

In order to explore the most important types of features in one go, we are going to focus only on the *building* data [2] (represented by nodes, ways and relations) in the one hand, and on the *road network* data [6] (represented by ways and relations) in the other hand. The study of Points of Interest (POI) is deprecated, since it is not interesting for the NLS.

From now on, the building data and the road network data are referenced as *buildings* and *roads* respectively.

1.4 Some related work

As of now, few massive contributions on our selected themes are documented on the OSM wiki [3]. Each import has also its own tagging rules, without any standardized labelling. Then, our work is being meant to allow identification of the massive contributions which were not properly documented.

In the same spirit as said in the Introduction, some Italian researchers shed light on the pros and cons of integrating OSM data in the process of producing governmental data sets [26], touching the discussion where in the future the FGI need to think if OSM could be an important information source for the NLS topographic database.

Chapter 2
Data retrieval from the OSM database

2.1 QuickOSM QGIS plugin

One very quick way to download the OSM data we are interested in here is to use the QuickOSM QGIS plugin. Based on the Overpass API [11], it features the "Quick query" tool, allowing the user to retrieve data on keys/values filtering (tags).

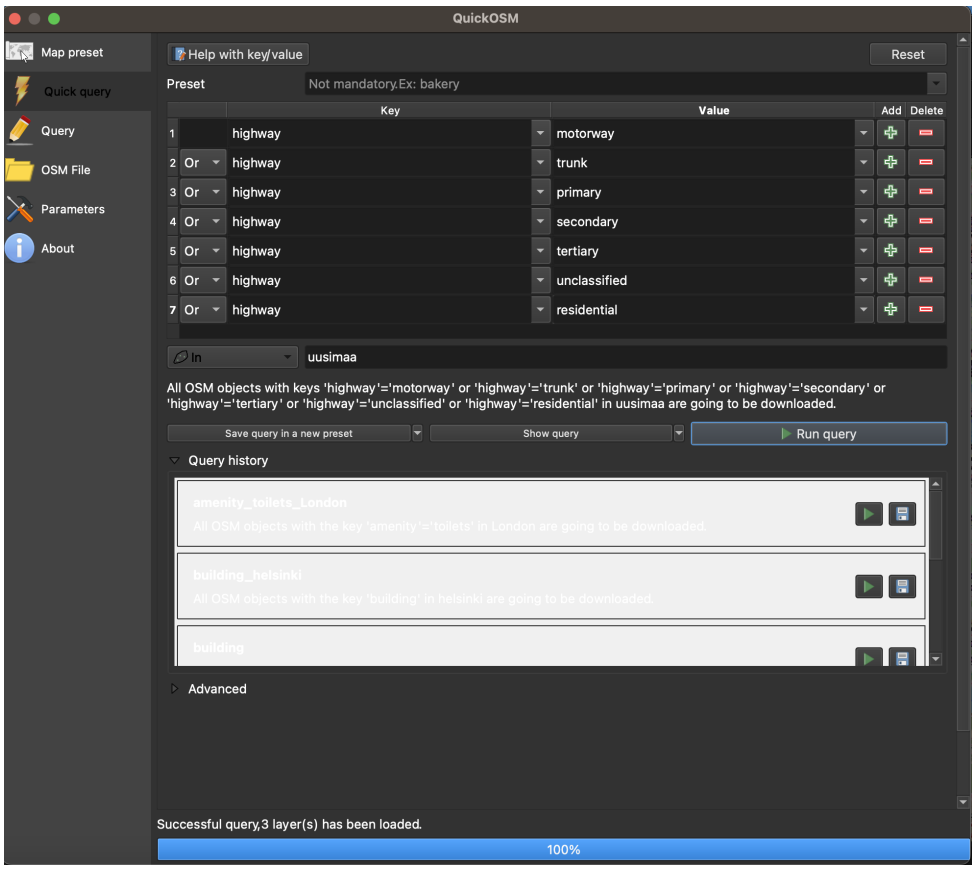


Figure 2.1: QuickOSM usage example in QGIS — *roads* from Uusimaa retrieval

As shown in Figure 2.1 above, several matching conditions are provided to get the desired output. In fact, simply putting the relevant key isn't enough. For example, *roads* features in OSM are identified with the *highway* key, but if no value is entered, the query doesn't return anything.

Then, this constraint force us to know the exact values that describe our *roads*, which is pretty irrelevant in our case: documentations on the OSM wiki [6] [2] don't specify clear boundaries for the *roads* tagging, same applying for the *buildings*'s. In the other hand, those queries don't give the upper hand on the data structure in the output, i.e. there is no control on the features tags and the

types of geometries. Some testing highlighted the missing of the source tag, which is indeed vital for our investigation.

Proceeding with this method does not guarantee to get all the wanted data, since there will always be some missing!

Eventually, the running time for loading the resulting attribute table in QGIS is quite long for the size of the actual data (only 69,207 features filtered by the query in Figure 2.1).

2.2 Geofabrik OSM data extracts provider

Following the previous discoveries, an alternative way to get our data is to download it directly from source. Geofabrik.de is a third party providing geographical extracts from the complete OSM database with up-to-date worldwide coverage.

Two downloading servers are available:

- *Public server*: contains regional data stripped from OSM contributors personal metadata (user names, user IDs, changeset IDs), as they are subject to data protection regulations in the European Union
- *Non-Public server*: reserved for OSM contributors and internal purposes, contains extracts with full metadata

With the permission of the Internship supervisor, **we will continue our study with data from the Non-Public server**, since *changeset IDs* are also required to identify NLS features (see section 3.2).

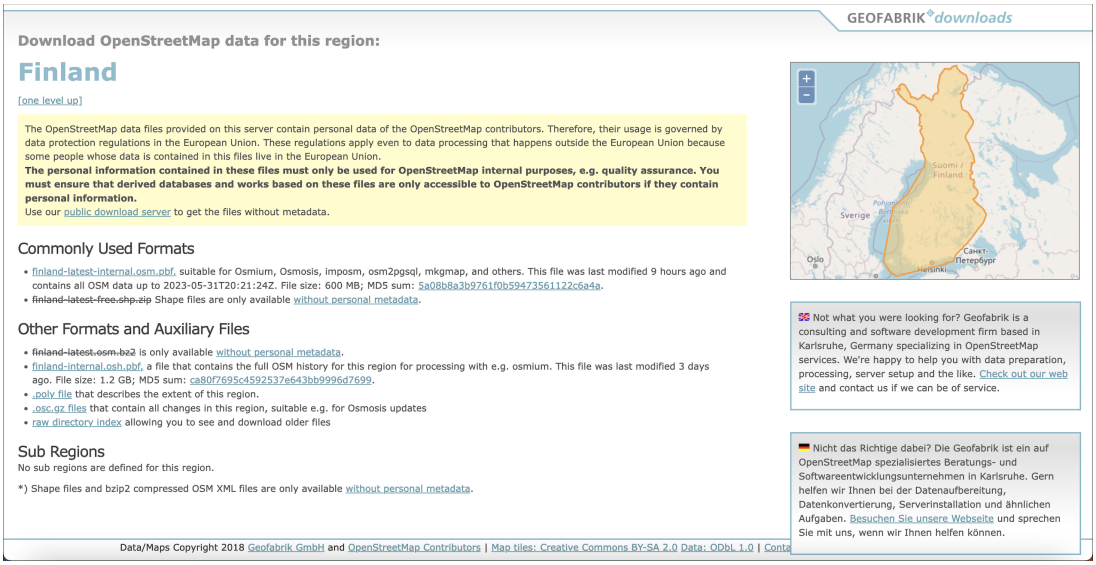


Figure 2.2: Finland's Non-Public data Geofabrik downloading webpage — URL: https://osm-internal.download.geofabrik.de/europe/finland.html

From this page, **`finland-latest-internal.osm.pbf` and `finland-internal.osh.pbf` files were downloaded on the 1st of June 2023 at 2pm**. The former is a *data file* containing OSM

data from a specific point in time (so as of the 1st of June here) *with at most one version per object (node, way, or relation)*. The second is a *history file* containing not only the current version of one object, but also all its history, i.e. there are zero or more versions for any object in this file.

Note that `.pbf` extension means the relevant file is in binary format. It can be read by most programs dealing with OSM data but not by a simple text editor. As for `.osm` or `.osh` extensions, these correspond to variants of the classical XML format. For more information on Geofabrik's data extracts, please see [14].

As you've maybe noticed in Figure 2.2, no subregions are available for Finland, i.e. data related to Uusimaa region is contained in our bigger file. Extracting sub sets from this downloaded data is the matter in the section 3.1.

2.3 Planet OSM

Planet OSM is a benchmark source allowing to download the whole OpenStreetMap database as a regularly-updated single file. Besides traditional `.pbf` and XML files, the website also offers a *changesets* file [4] containing complete metadata, for all the changes that ever happened in OpenStreetMap's history.

Since this file wasn't available on the Geofabrik's servers, it was downloaded from Planet OSM (**on 05/06/2023 at 5pm**) and we'll be of use later, especially for the collection of features modifications in the section 3.3.

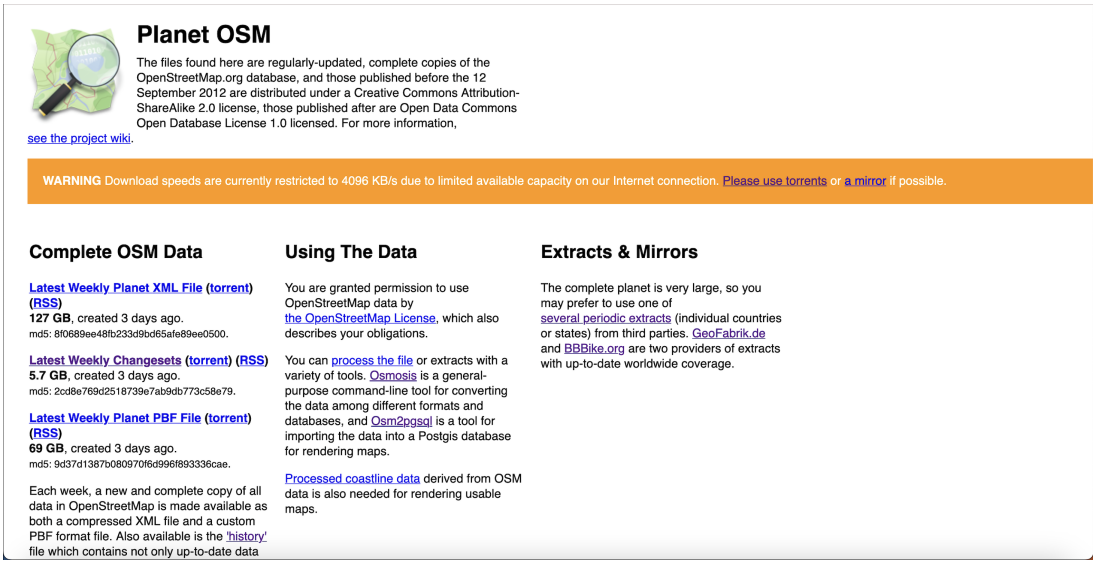


Figure 2.3: Planet OSM website homepage — URL: https://planet.openstreetmap.org/

Chapter 3
Data manipulations

3.1 Sub sets extraction from raw downloaded data files

As announced in chapter 1, our study will be limited exclusively to *buildings* and *roads* from Uusimaa region. In order to get those sub sets data from downloaded data, *Osmium Tool* we'll be used.

3.1.1 Osmium Tool

Osmium Tool [24] is a command-line tool based on the C++ library libosmium [22] and designed for processing OSM data. It features plenty of useful functions [23], as well as fast-reading properties, and often outperform similar tools.

Considering the unique shape of OSM data, the decision was made to carry on with this tool for the data extracting phase instead of QGIS, as it allows a greater flexibility, even so the learning curve is pretty steep in the early ages.

3.1.2 Data & History files

The data slicing was done in 2 stages: restricting the area covered by the data to the Uusimaa region, then extracting the themes of interest, i.e. *buildings* and *roads*.

Getting data from Uusimaa: we use the `osmium extract` command for creating geographical extracts from an OSM file. The geographical extent for the clip was given as a *bounding box* around Uusimaa, which includes a little more of neighboring regions to avoid edge effects.

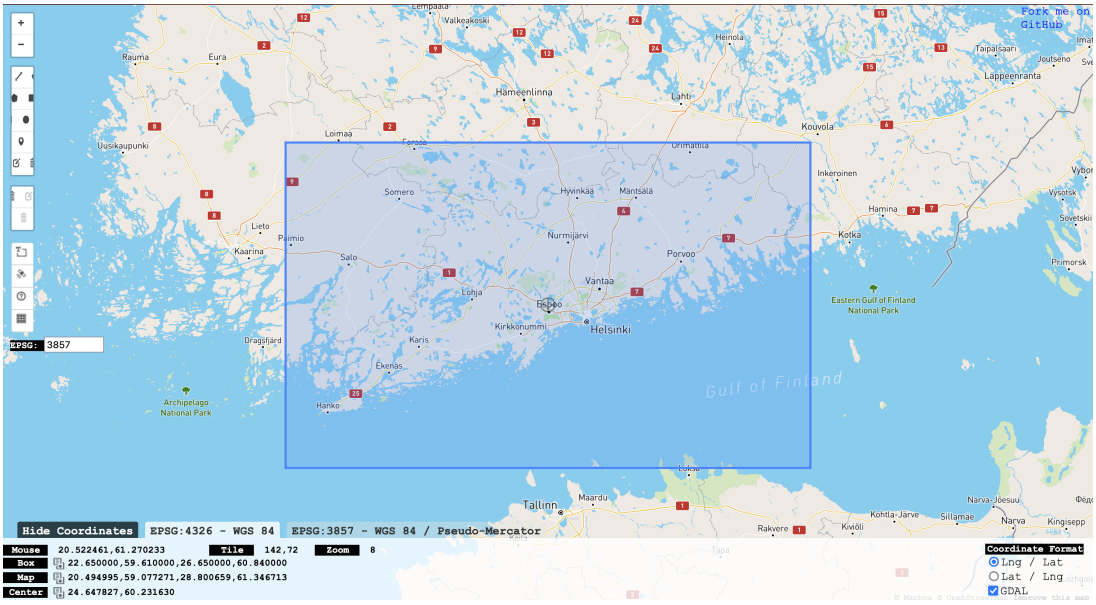


Figure 3.1: Bounding box used for Uusimaa's data clipping, lng/lat coordinates (22.65,59.61,26.65,60.84) — Uusimaa boundaries in *light gray* inside the blue rectangle (credits: bboxfinder.com)

Here's what the final command looks like:

```
1 osmium extract -b 22.65,59.61,26.65,60.84 finland-latest-internal
    .osm.pbf -o uusimaa.osm.pbf
```

The result from this code launch is **`uusima.osm.pbf`**, a new binary file containing all the features in the Uusimaa region.

We proceed in much the same way for the history file, except that we add the `-H` option to make the command compatible with this particular type of file: in fact, most programs using OSM data expect features IDs to be unique, so they can't work with history data! As a result, **`uusimaa-history.osh.pbf`** is also created.

Getting *buildings* and *roads* from Uusimaa **(only for the data file)**: we use the `osmium tags-filter` command for filtering objects matching specified keys/tags with the previous `uusimaa.osm.pbf` file as the input. Here, unlike in section 2.1, the filtering doesn't require specific values to be matched with the keys.

The process is then greatly simplified. Here's the command used for the *roads* extraction:

```
1     osmium tags-filter uusimaa.osm.pbf wr/highway -o roads-
        uusimaa.osm.pbf
```

Just above, we looked for all ways and relations (`wr/`) that have been tagged with the `highway` key, which identifies the *roads* features.

The same operation is realized to get the *buildings* data:

```
1     osmium tags-filter uusimaa.osm.pbf building -o buildings-
        uusimaa.osm.pbf
```

Here, geometry types are not specified, since we are interested in all of them (nodes, ways and relations).

At the end of this step, **roads-uusimaa.osm.pbf** & **buildings-uusimaa.osm.pbf** are created, corresponding to Uusimaa road network features and Uusimaa buildings features respectively.

3.1.3 Changesets file

Likewise, a bounding box is again used here to cut Uusimaa's changesets from whole planet's, but with the specific changeset command `osmium changeset-filter`.

```
1  osmium changeset-filter -B 22.65,59.61,26.65,60.84 -o uusimaa
       -changesets.osm.bz2 changesets-230529.osm.bz2
```

The output is **uusimaa-changesets.osm.bz2**, a bzip2-compressed XML file.

3.2 Source tags values retrieval

3.2.1 Strategy

As introduced in section 1.3, the key for browsing NLS features is based on the source tag. Historically, it was the usual practice for adding this source information as an attribute on each relevant feature. However with modern editors, the source tag is more and more put on the changeset where several features edited in the same session are regrouped [9].

The aim of the following paragraphs is then to get the features' sources by looking at the *feature's attributes level*, but also at the *changesets level* in which the features are included.

3.2.2 Python-based method

The first idea was to take a look at our extracted data in QGIS (**roads-uusimaa.osm.pbf** & **buildings-uusimaa.osm.pbf**). Despite QGIS's ability to handle `.pbf` files by adding it as a Vector Layer, the files' sizes were way too big for the program to load correctly. Then, one solution was to convert our files into *SpatiaLite* databases using *GDAL ogr2ogr command-line tools*, and connect those to QGIS [1]:

```
1  ogr2ogr -f SQLite -dsco SPATIALITE=YES buildings-uusimaa.db
       buildings-uusimaa.osm.pbf
```

Here is an example for *buildings*, resulting in a `.db` file, readable in QGIS through a SpatiaLite connection.

Unfortunately, an attribute analysis in QGIS of the features in our `.db` files again revealed the absence of the source tag in our data, thus forcing our investigation to focus on changesets.

Then, the imagined processing chain is based on the *changeset id*, a unique identifier for each changeset entity. Also considering that every feature in OSM is associated to a changeset, this link allows source information on changeset to be assigned to each feature. To do so, we use a homemade Python script which generates a `.csv` file on an initial data file input, containing the following information (fields):

- `type`: type of feature (node, way or relation)
- `id`: OSM unique identifier for the feature
- `changesetId`: OSM unique identifier for the associated changeset
- `source`: source information about the changeset

The script is based on OSMCha [10] (short for OpenStreetMap Changeset Analyzer) Python library used for advanced changesets analysis, and PyOsmium [18] which is an adaptation of the previous libosmium C++ library for Python support, used here for OSM file reading. For more details, please check the relevant code and the preliminary results overview both available in appendix A.

Note that the `.csv` output format is justified as this will allow an easier import into a PostgreSQL database for future string-targetted queries on the source spelling (i.e. references to NLS).

However, the proper functioning of this code depends on the input file's size. In fact, the source information retrieval corresponds to single requests to the OSM server, meaning the number of requests is equal to the number of the input features. This way of proceeding is not acceptable for our actual files considering its sizes, as it will also lead to Internet network saturation with an endless code launching.

This method is eventually deprecated.

3.2.3 GDAL Configuration file for OSM import

Thanks to a hint from the internship supervisor, **the feature-level source tag has finally been explicated by modifying the `osmconf.ini` file**. It is a configuration text file delivered with the GDAL installation packages, allowing to add the exact attributes wanted to be appearing in the output file generated from a ogr2ogr command (configuration file content available in appendix B).

In our case, 2 additional tags are considered:

- `source`: the feature-level source information
- `created_by` [8]: information on the software used for the feature edition

In addition to potential NLS-tagged features, the last tag allows broader investigations regarding features which might have a link to the NLS in case they are wrongly tagged for instance. In the same spirit, the `converted_by` tag could have been interesting, but has no known use according to the OSM wiki [7].

Also, in anticipation of the very next step, the field `osm_changeset` has been modified to *yes* value in the configuration file, enabling the changeset ids display.

Eventually, we use ogr2ogr commands to convert our data file from one format to an another, in our case from `.pbf` to `.gpkg` (Geopackage file) for QGIS reading. Here is an example for the *buildings* data from Uusimaa:

```
1    ogr2ogr -f GPKG buildings-uusimaa.osm.gpkg buildings-uusimaa.
       osm.pbf
```

Attribute table verifications in QGIS have shown the correct addition of the desired attributes, which will be the subject of more advanced analysis in the next chapter.

3.2.4 PostgreSQL method

Following the previous discoveries, another way for getting changeset-level source tag is to look at our *Uusimaa's changesets file* directly. The idea is to import this file into a PostgreSQL database, and then to make SQL JOIN operations with imported feature tables on the changeset id. The link is pretty much the same as in subsection 3.2.2, but now the data fetching will be done *locally* instead.

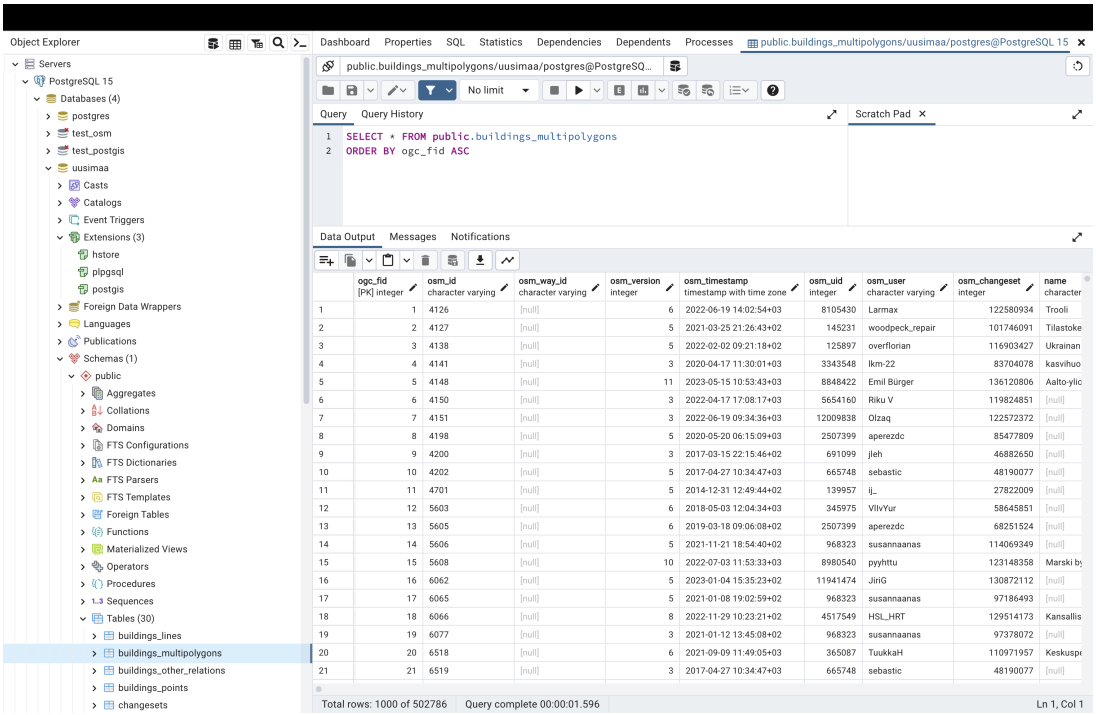


Figure 3.2: *pgAdmin 4* interface showing contents from the `uusimaa` database

The *uusimaa* database was created under PostgreSQL 15, with *postgis* and *hstore* extensions. The former allows storing features' geometry information, while the latter is a special data type for storing different information of undefined length as sets of key/value pairs within a single column [15]. It will host the feature data tables as well as the changesets table.

For importing the feature data, there are several possibilities, and the first one is to use *Osm2pgsql* tool. It is command-line software especially designed for OSM imports into PostgreSQL/PostGIS databases. However, as documentations state, getting to grips with Osm2pgsql is rather difficult for beginners [21]. No significant results were obtained after several attempts.

An alternative to Osm2pgsql can be found in the GDAL library. In fact, GDAL features an *OSM Vector driver* [13] which can also provide supports for OSM file import, but in a much simpler way. Through a single-line bash command, this driver imports any `.pbf` data file by splitting it into 5 distinct layers/tables (*points*, *lines*, *multilinestrings*, *multipolygons* and *other_relations*), while relying on custom parameters supplied by the user in the `osmconf.ini` configuration file as in section 3.2.3. Here is an example for importing *buildings* from Uusimaa in our database:

```
ogr2ogr -f PostgreSQL "PG:dbname='test_osm' host='localhost'
    port='5432' user='postgres' password='postgres'" buildings-
    uusimaa.osm.pbf -lco COLUMN_TYPES=other_tags=hstore
```

In fact, the actual command template used to import our data contains an additional option, designed to allow imports into a single database, by assigning a unique name to each table imported from a data file. No better solutions were found but to repeat the following code 10 times in total (5 times on each data file for its 5 different layers):

```
ogr2ogr -f PostgreSQL "PG:dbname='uusimaa' host='localhost'
    port='5432' user='postgres' password='postgres'" buildings-
    uusimaa.osm.pbf multipolygons -nln buildings_multipolygons
    -lco COLUMN_TYPES=other_tags=hstore
```

This was an example for importing the *multipolygons* layer of the *buildings* from Uusimaa in our database.

At the end of the import, some tables were empty and were therefore deleted, resulting in a final number of **7 feature data tables** (see appendix C for UML representation).

As for the changesets, they were imported using a Python script (see appendix D) based on the ElementTree XML API [25] for parsing XML data, as well as on the Psycopg library [16], which lets you access a PostgreSQL database from Python. For this to work properly, the creation of a table with a very specific structure is required beforehand, in order to receive incoming data as shown in Figure 3.3 below:

```
CREATE TABLE public.changesets
(
    changeset_id integer NOT NULL,
    created_at timestamp without time zone,
    closed_at timestamp without time zone,
    open boolean,
    "user" text,
    uid integer,
    min_lat double precision,
    min_lon double precision,
    max_lat double precision,
    max_lon double precision,
    num_changes integer,
    comments_count integer,
    other_tags hstore,
    PRIMARY KEY (changeset_id)
);

ALTER TABLE IF EXISTS public.changesets
    OWNER to postgres;
```

Figure 3.3: `changesets` table creation SQL code

Note that a particular attention was given to the changesets' tags, where the source information is stored. Since there is not a fixed number of tags on any changeset, all of them were put in a single column using the `hstore` data type.

Eventually there are still JOIN operations to make, but they'll be of interest in the very next chapter, alongside SQL queries for NLS-like strings pattern matching.

3.3 Feature-level modifications collecting

Although it was originally planned to carry out a full analysis of the changes as done in [19], it was decided to concentrate on the *number of geometric modifications* that the features have undergone since their integration into OSM, which is more interesting and useful from the point of view of the NLS.

3.3.1 Intensity rather than quality

This section's work does not measure the magnitude of how a given OSM feature has changed. In fact, a big geometry change will be treated just as a small one, but instead the willingness of the contributors to make changes to the OSM database is assessed.

3.3.2 About the changesets

The natural way for studying the modifications as described in [19] is to look into the *changesets* which were introduced in section 2.3. To be more precise, any changeset is supposed to regroup all the changes one contributor can make during an editing session, i.e. from the moment he opens his editor to the one he commits his changes to the OSM server [4]. In reality, changesets don't provide targeted information on modified features, but only general information about the scope of the changes (opening/closing timestamps of the changeset, geographical extent of the changes, overall number of modifications, etc.), while being referenced at the level of each feature with the *changeset id* to ensure a proper logging.

3.3.3 Digging into the OSM history file

The OSM history file is more promising, as it contains concrete data about all the versions that have ever existed for any OSM feature, thus allowing comparison between the versions to monitor the changes throughout time.

As stated in section 3.1.2, little programs are able to deal with OSM history files. The content of our history file from Uusimaa (`uusimaa-history.osh`) was then imported into the *uusimaa* database as `history` table using a custom Python script (cf. Appendix H), based on the *lxml* XML parsing Python library [12].

Superior to the native ElementTree API (used in 3.2.4 for the changesets import) in terms of processing time and functionalities, lxml allows an iterative approach, meaning the XML-based history file is read on the fly, thus reducing the memory usage considerably. To free up even more memory, intelligent variable management was put in place with outdated variables' deletion at each iteration. This constant attention to memory management stems from the fact that initial tests were based on storing the entire XML file into memory before reading it. This strategy do work for the history import of small areas like the Otaniemi neighborhood (famous R&D district, where the FGI is located) within a matter of a few seconds, but completely overwhelm the Python kernel capacities with a way bigger region as Uusimaa's (which account for 223.2 MB in the raw binary format, but for 7.73 GB when converted into XML!). For more information, the former code implementing this latter naive approach is available in Appendix I.

Once the history is imported, it needs to be analyzed to locate the geometric modifications. This task was done using again a Python script (cf. Appendix J), which core of its operation is based on the `geometry_modification` recursive function (from line 131 in the code). Due to the OSM data structure, geometric information is exclusively attached to the *node* elements (see *OSM data structure* in section 1.2). As a result, references need to be resolved in order to access

geometric information on *ways* and *relations*, which is handled by the recursive behavior of the `geometry_modification` function, while the `geometry_references_resolver` function offers a *version resolving system* suggestion, as referenced members of ways and relations might also have several versions to consider.

Chapter 4
Data analysis and Visualisations

4.1 Identifying the massive contributions of NLS data in OSM

4.1.1 NLS references at the feature-level source tag

Following the highlighting of the feature-level source tag field in the section 3.2.3, it should now be possible to identify the NLS data. However, as mentioned at the beginning of our study in 1.4, few massive contributions from the NLS have official documentation, adding that there is not a particular set of source tag values provided by the NLS for us to search the exact information we want.

It was therefore necessary to search blindly, considering for example particular values that the source tag would be likely to take to identify the data coming from the NLS. A first NLS source typology was then designed as shown in Figure 4.1, including the two official languages of Finland which are Finnish and Swedish, but also English, this with a view to taking into account as much data as possible.

```
"source" ILIKE '%mml%' OR
"source" ILIKE '%nls%' OR
"source" ILIKE '%maanmittauslaitos%' OR
"source" ILIKE '%national land survey of finland%' OR
"source" ILIKE '%finnish geospatial research institute%' OR
"source" ILIKE '%fgi%' OR
"source" ILIKE '%Paikkatietokeskus%' OR
"source" ILIKE '%lmv%' OR
"source" ILIKE '%Lantmäteriverket%' OR
"source" ILIKE '%Geodatacentralen%'
```

Figure 4.1: First NLS source tag value typology

These values, framed by '%' signs, correspond in fact to patterns, which we search for within the values of the source tags. The Figure 4.2 below shows the results we get for both *buildings* and *roads* data sets, by counting the features which fall under our typology.

Overall, as we can see, the vast majority of features from the 2 datasets have no source information (i.e. source tag left blank). The percentages for NLS data are thus very low, while further tests have shown that also considering *created_by* and *other_tags* tags (introduced in section 3.2.3, with the latter being a tag for everything as described in [13]) doesn't bring much better numbers, with a slight variation of a few dozen features at most (see Appendix E for related numbers).

Sources / Type of features	Points (49,353 features)	Lines (85 features)	Multipolygons (502,786 features)	All features (total)
NLS	0.008% *(4 features)*	0% *(0 features)*	6.6% *(33,149 features)*	6 %
Other	0.7% *(363 features)*	11.8% *(10 features)*	6.1% *(30,576 features)*	5.6%
No source	99.3% *(48,987 features)*	88.2% (75 features)	87.3% *(439,061 features)*	88.4 %

Table 1 — Source key values proportions for the "buildings" dataset (552,224 features)

Sources / Type of features	Points (105,452 features)	Lines (468,982 features)	Multipolygons (4,214 features)	All features (total)
NLS	0.2% *(199 features)*	0.7% *(3,410 features)*	0.2% *(9 features)*	0.6%
Other	3.5% *(3,718 features)*	5.6% *(26,092 features)*	2% *(86 features)*	5.2%
No source	96.3% (101,535 features)	93.7% *(439,480 features)*	97.8% *(4,119 features)*	94.2%

Table 2 — Source key values proportions for the "roads" dataset (578,648 features)

Figure 4.2: Feature-level source tag values proportions

4.1.2 Adding changeset-level source tag information

Deeper investigation is then needed on the changesets, where the most source information should be contained. From the 7 feature tables and the `changesets` table created in the section 3.2.4, 3 classes of new tables are defined:

- feature tables beginning with '*nls*': contain features falling under the typology introduced in the section 4.1.1 and including matching source information from the changesets
- feature tables beginning with '*no_source*': contain features with both feature-level and changeset-level source tag empty
- feature tables beginning with '*other*': contain features that do not fall into any of the categories listed above

As a result, 21 additional tables are added to the *uusimaa database*, mostly created by SQL LEFT JOIN operations and string pattern analysis, so to allow quick access to relevant data without complex querying if needed in the future. For more insights on the SQL queries used to build those tables, please take a look at Appendix F.

Here are then the updated results from Figure 4.2 by including additional source information from the changesets:

The inclusion of sources related to changesets seems to be bearing fruit, with overall about *3.6 times more* NLS buildings being taken into account (6% previously), and about *16.3 times more*

Sources / Type of features	Points (49,353 features)	Lines (85 features)	Multipolygons (502,786 features)	All features (total)
NLS	2.6% *(1,305 features)*	2.4% *(2 features)*	23.6% *(118,810 features)*	**21.8%**
Other	35.1% *(17,300 features)*	68.2% *(58 features)*	27.6% *(138,453 features)*	28.2%
No source	62.3% *(30,748 features)*	29.4% *(25 features)*	48.8% *(245,523 features)*	50 %

Table 5 — Source key values proportions for the “buildings” dataset (552,224 features)

Sources / Type of features	Points (105,452 features)	Lines (468,982 features)	Multipolygons (4,214 features)	All features (total)
NLS	4.5% *(4,692 features)*	11.1% *(51,861 features)*	4.8% *(201 features)*	**9.8%**
Other	48.5% (51,168 features)	47.1% *(221,146 features)*	55.4% *(2,335 features)*	47.5%
No source	47% *(49,592 features)*	41.8% *(195,975 features)*	39.8% *(1,678 features)*	42.7%

Table 6 — Source key values proportions for the “roads” dataset (578,648 features)

Figure 4.3: Complete source tag values proportions

NLS roads being considered (0.6% before). As more than the half of the features from both data sets have the source tag filled, we consider these numbers as sufficient to continue our study.

4.1.3 Focus on NLS-like source values spelling

A more detailed analysis on the source values spelling for features being categorized as *NLS* can also be interesting. The 20 most recurrent values from the `multipolygons` layer of the *buildings* data set are shown below in Figure 4.4.

	source character varying	occurences numeric
1	MML	37229
2	MML/NLS	17648
3	aerial imagery;MML background map	12734
4	MML2017	10322
5	MML Orthophoto	8739
6	MML Topographic Map	8601
7	NLS.fi	4649
8	MML2016	4335
9	MML_2012	2524
10	local knowledge, aerial, MML	2010
11	aerial imagery;MML Ortophoto	1937
12	mml2018/helsingin karttapalvelu.	1925
13	Mapbox Satellite / National Land Survey of Finland	1788
14	MML2018	1261
15	NLSF	1151
16	MML2019/Helsingin kiinteistökartta	1121
17	MML Background Map	959
18	Addresses: MML/DVV.fi; aerial imagery; map imagery;	945
19	MML Background Map; MML Orthophoto; MML Topographic Map	799
20	MML Orthophoto; MML INSPIRE Buildings Maastotietokanta (WMS)	690

Figure 4.4: Source tag value most prominent occurences from `nls_buildings_multipolygons`

As we can see, the NLS can be mentioned in numerous ways, with the 'MML' value being the most used without big surprise. However, a significant number of values are being composed of sub-values in accordance with [9], but actually showing the complexity in identifying proper NLS data. Indeed, the Figure 4.4 provide us with just an overview on how NLS is mentioned in OSM, as more than 400 additional different values have been filtered by our SQL query (cf. Appendix G).

From that, **it is impossible to say if one feature is really coming from the NLS**, as no official documentation related to NLS features tagging in OSM exist at this moment to confirm our assumptions. Then, it also supports the fact that the OpenStreetMap project is anarchic with no real constraints on data contributions, in contrary to authoritative data providers such as National agencies, where a lot of standards govern the formatting of data.

As a result, the following paragraphs are intended to provide some insights into features that have a high probability of belonging to imports from the NLS, but with no absolute certainty that this is the case. For the sake of brevity and misuse of language, the features concerned will be referred to as *NLS*'s.

4.1.4 NLS massive contributions visualizations

From the identified NLS features in 4.1.2, it is now possible to highlight areas of massive contributions through a spatial pattern analysis, which relies on NLS feature proportions in different geographical locations. This can be achieved with the use of hexagon grids.

A hexagon grid (created using MMQGIS QGIS Plugin [20]) is defined by two parameters:

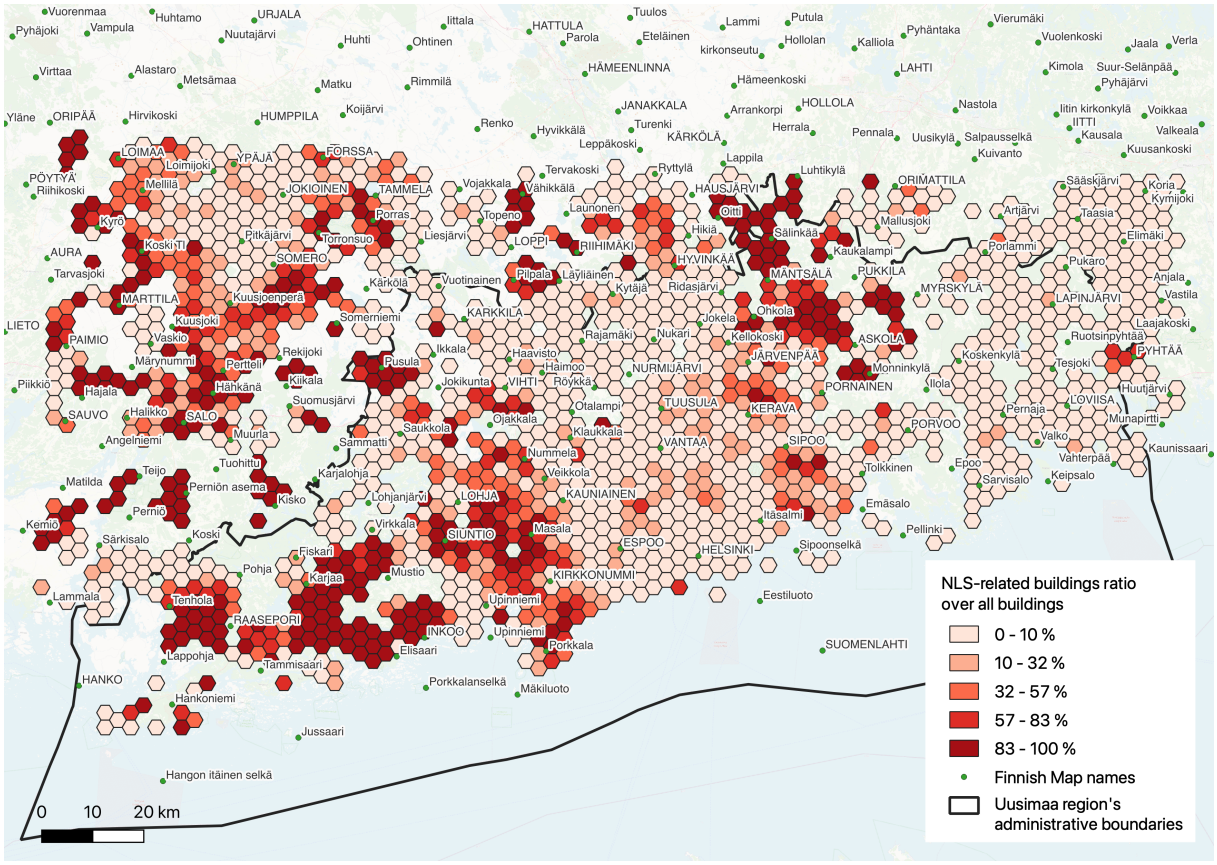
- *the extent of the grid*: covered area by the hexagons
- *the grid interval*: hexagon size

The last parameter is of particular importance: the hexagons shouldn't be too small or too big, with in the former case the result being similar to a simple plot of the data, while in the latter the hexagon size prompts the lose of spatial patterns. In order to take this into account, we'll proceed with an hexagon grid having $2 * apothem = 2500\,m$ (ETRS89 / TM35FIN(E,N) [EPSG:3067] CRS Units) as *grid interval*, covering a little more than Uusimaa region's extent.

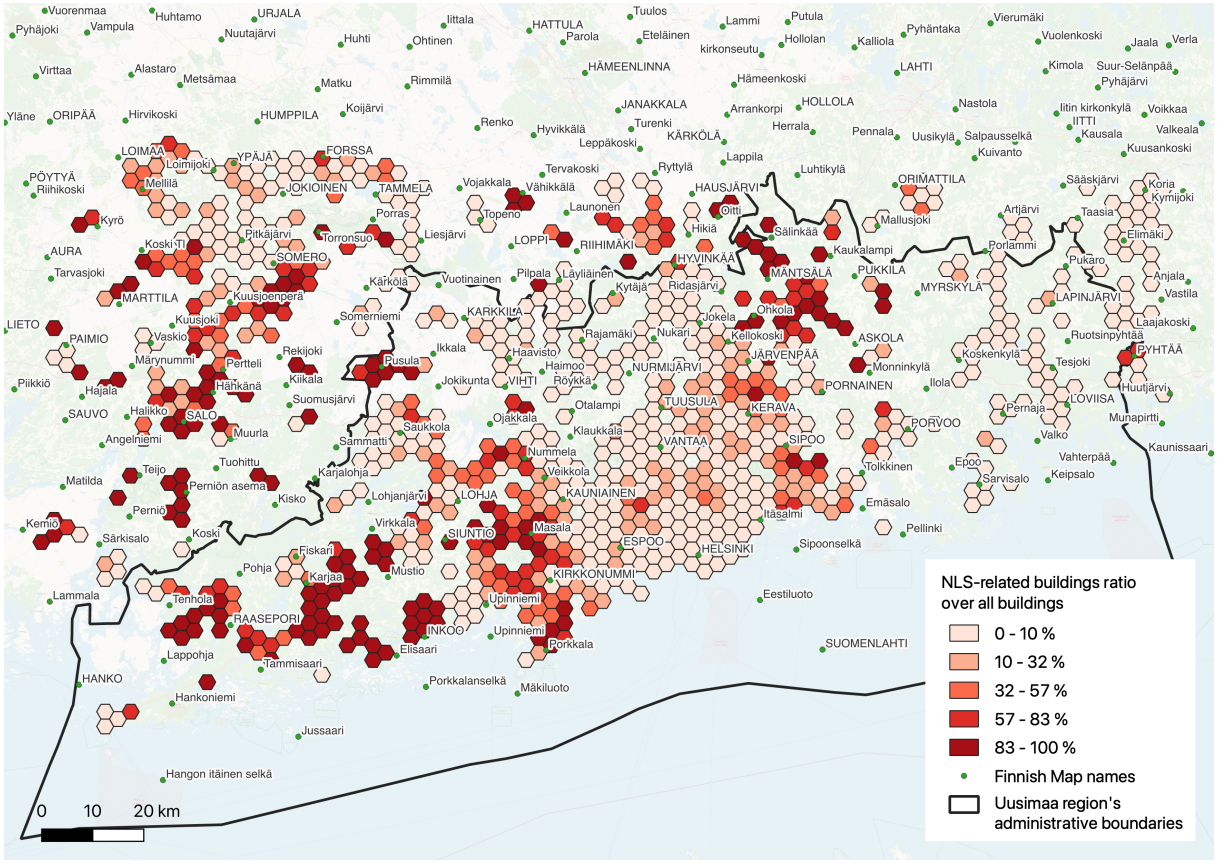
Then, by counting the number of centroids of each polygon from both `nls_buildings_multipolygons` and `buildings_multipolygons` for example inside any hexagon of the grid, the NLS-related buildings ratio over all buildings can be calculated for any hexagon with the following formula:

$$\text{nls\_ratio} = \frac{\text{nb of } \texttt{nls\_buildings\_multipolygons} \text{ centroids}}{\text{nb of } \texttt{buildings\_multipolygons} \text{ centroids}} * 100$$

One downside of this way of thinking, is that if an hexagon of the grid was to contain only a unique building in its area with this building appearing to be tagged as NLS, then nls_ratio $= 100\%$, which is completely wrong for our purpose! In fact, the goal here is to identify aggregated groups of features as hexagons. Hence, the *nb of `buildings_multipolygons` centroids* should be the greatest as possible, as it is an indicator of the size of what we actually consider as a massive contribution. Several map configurations were tested with an ascending minimum number of buildings per hexagon in the Figure 4.5 below.

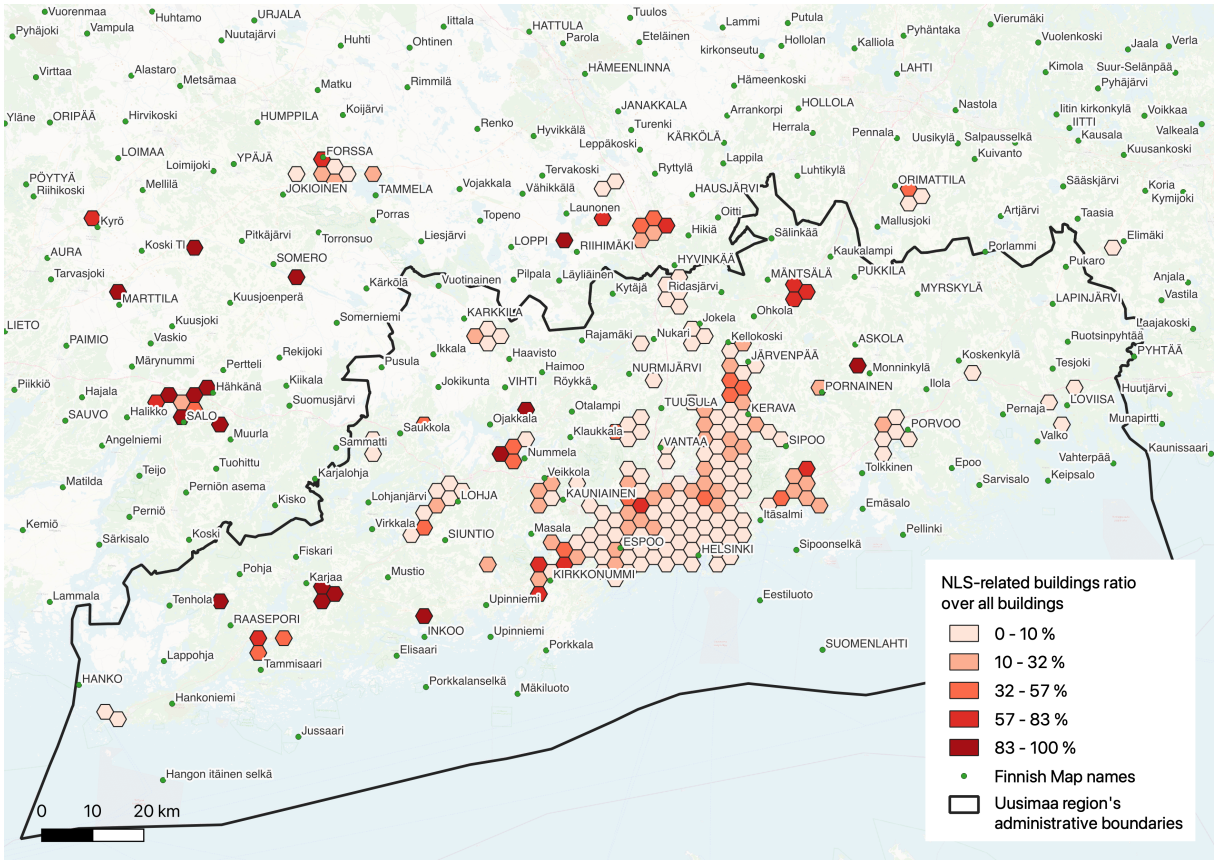


(a) $\forall hexagon \in grid, min(nb_{\text{buildings}}) = 20$



(b) $\forall hexagon \in grid, min(nb_{\text{buildings}}) = 100$

Figure 4.5: Hex maps on NLS-related buildings proportion over all buildings (Map base credits: © OpenStreetMap contributors)



(c) $\forall hexagon \in grid, min(nb_{\text{buildings}}) = 500$

Figure 4.5: Hex maps on NLS-related buildings proportion over all buildings (cont.)

Right from the outset, we note that the density of the grid is inversely proportional to the minimum number of buildings per hexagon. This behavior is consistent, since our selection criterion on the size of massive contributions becomes increasingly restrictive. Then, for a fixed minimum number of buildings per hexagon (i.e. if we focus on one map), hexagons are associated with several percentage categories. The higher the percentage, the higher the rate of NLS buildings in a hexagon, meaning that our massive contributions are concentrated in the upper half of the percentages: the higher the percentage, the more likely it is that the buildings in the hexagon under consideration belong to a massive contribution from NLS.

Indeed, as we've seen it in the previous subsection 4.1.3, it is impossible to determine with certainty the massive contributions from NLS, but this study may support further research in the future. Still, by considering the fifth with the highest ratios from those first results, we can conclude that NLS massive contributions of buildings are likely for the most to be located in the west of the Uusimaa region.

On the other hand, even if the number of buildings per hexagon changes, we can see that the percentages around the Capital Region remain low. One hypothesis for this is that the data in this area comes from the relevant municipalities, from which better quality material than NLS's is available.

4.2 Feature-level modifications underwent by NLS data since its import into OSM

By applying the method described in section 3.3.3, the following numbers in Figure 4.6 were obtained for the NLS-related buildings from Uusimaa.

```
In [3]: runfile('/Users/alexysren/Desktop/test3/nls_buildings_multipolygons_geometry_modif/typology_modif_encoding_copy.py', wdir='/Users/alexysren/Desktop/test3/
nls_buildings_multipolygons_geometry_modif')
100%|██████████| 118810/118810 [12:25<00:00, 159.45it/s]


---------------  Geometry modification statistics ---------------

Feature count: 118810
Among the  118810 features
   - 8240 underwent geometric modifications  ( 6.9 % )
   - 110555 have not been modified  ( 93.1 % )
   - 15 could not be analysed  ( 0.0 % )
*    Average number of geometry modifications per feature: 0.1
*    Max number of geometry modifications: 24.0


In [4]:
```

Figure 4.6: Statistics over the number of *geometry modification* for `nls_buildings_multipolygons`

While a negligible number of features encountered some processing errors with the `typology_modif_encoding_copy.py` code launching, the vast majority of the NLS-related buildings from Uusimaa have not been modified since their import. Overall, the features have also undergone very few changes, but modified features appear to account for a significant proportion in absolute terms, with just over 8,000 modified buildings. It would be worthwhile to work on locating these buildings (related Python code in Appendix K), most of which seem to be clustered around the Capital region as shown in Figure 4.7.

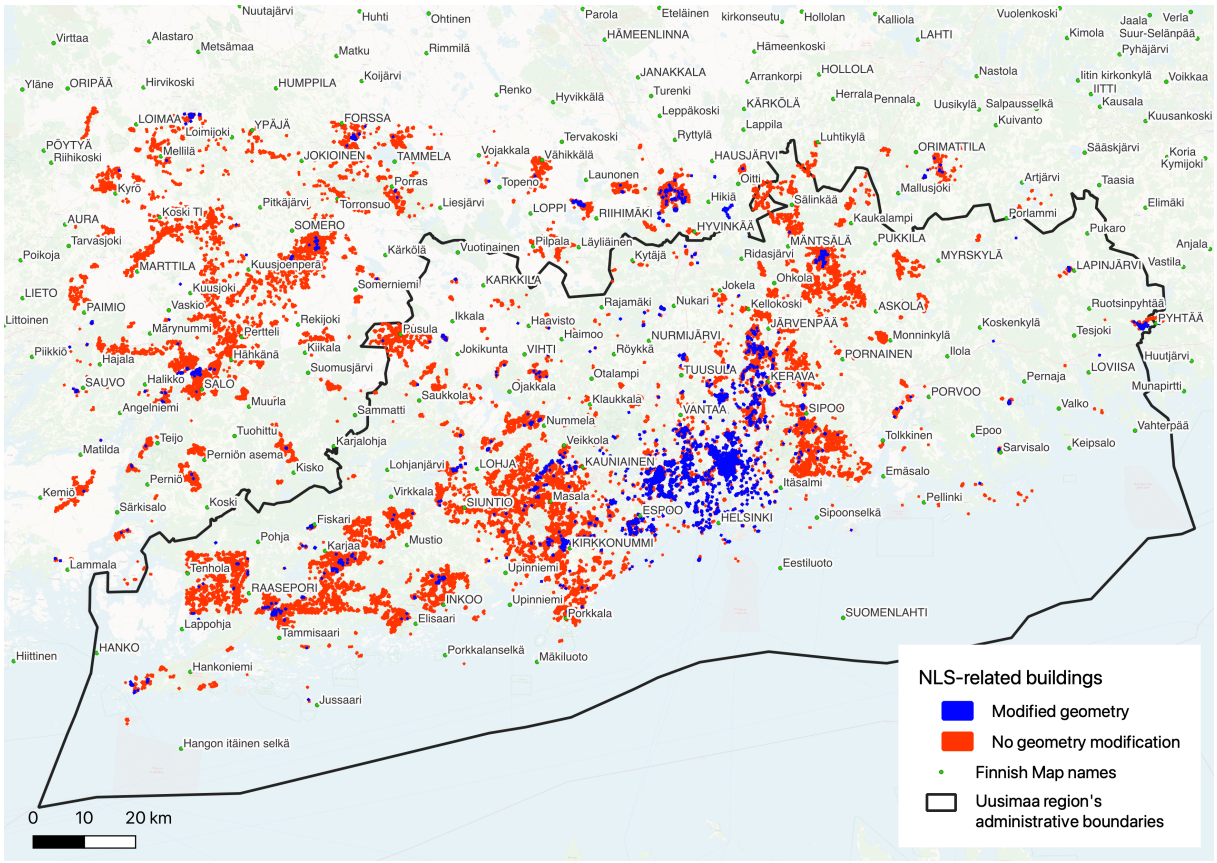


Figure 4.7: Spatial distribution of geometrically modified NLS-related buildings (Map base credits: © OpenStreetMap contributors)

Conclusion

Our study presents an approach to identify massive contributions from a national official agency, the NLS, to the OpenStreetMap database without clear documentation, and to make preliminary assessments of the contributions' intensity on the geometry of the targeted features. We then presented the context of the study, before downloading the data of interest from various data providers. New tools designed for OSM were also used to handle the raw data and transform it to fit the area of Uusimaa, and to retrieve the source key values from both features and changesets, while obtaining the geometric modifications was mainly done by creating our own custom tool using Python, due to the lack of software able to handle historical data from OSM.

Initial results show the importance of including changeset-level source information to obtain the most data of interest, while advanced analysis of the spelling of source values highlights the uncertain nature of our identification process when searching for NLS features. Some simulations were still possible and help to assume that a majority of buildings with massive contributions from the NLS are located in the west of the Uusimaa region. However, considering that in our best case scenario a little less than 50% of the features don't have the source tag filled, further research would be welcome in order to verify if some NLS-related features were not tagged and missed in our research.

Our final discoveries on the geometrically modified features provide insights on the location of the relevant NLS buildings which are of interest to OSM contributors, and can be the start of a deeper investigation on whether the quality of the modified features is acceptable for the NLS to benefit from it. With the emergence of AI-based solutions, these future results could also be compared to those provided by self-sufficient units.

Eventually, the current research should be extended nationwide across the whole of Finland, in order to see any differences or patterns on a larger scale. It may also be interesting to look at more data themes from which different conclusions can be drawn.

References

[1] GIS StackExchange Community. *Loading .osm.pbf file in QGIS*. StackExchange. 2019. URL: https://gis.stackexchange.com/questions/39112/loading-osm-pbf-file-in-qgis (visited on 06/01/2023).

[2] OpenStreetMap Community. *Buildings*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/Buildings (visited on 05/30/2023).

[3] OpenStreetMap Community. *Category:Import from Finland*. OpenStreetMap Wiki. 2013. URL: https://wiki.openstreetmap.org/wiki/Category:Import_from_Finland (visited on 06/28/2023).

[4] OpenStreetMap Community. *Changeset*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/Changeset (visited on 06/06/2023).

[5] OpenStreetMap Community. *Elements*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/Elements (visited on 06/12/2023).

[6] OpenStreetMap Community. *Highways*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/Highways (visited on 05/30/2023).

[7] OpenStreetMap Community. *Key:converted_by*. OpenStreetMap Wiki. 2022. URL: https://wiki.openstreetmap.org/wiki/Key:converted_by (visited on 06/12/2023).

[8] OpenStreetMap Community. *Key:created_by*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/Key:created_by (visited on 06/12/2023).

[9] OpenStreetMap Community. *Key:source*. OpenStreetMap Wiki. 2022. URL: https://wiki.openstreetmap.org/wiki/Key:source (visited on 06/06/2023).

[10] OpenStreetMap Community. *OSMCha*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/OSMCha (visited on 06/06/2023).

[11] OpenStreetMap Community. *Overpass API*. OpenStreetMap Wiki. 2023. URL: https://wiki.openstreetmap.org/wiki/Overpass_API (visited on 05/29/2023).

[12] lxml developers. *lxml - XML and HTML with Python*. 2022. URL: https://lxml.de/ (visited on 08/02/2023).

[13] gdal.org. *OSM - OpenStreetMap XML and PBF*. GDAL documentation. 2023. URL: https://gdal.org/drivers/vector/osm.html (visited on 06/15/2023).

[14] Geofabrik.de. *Data Extracts - Technical Details*. 2019. URL: https://osm-internal.download.geofabrik.de/technical.html (visited on 06/01/2023).

[15] Andrew Gierth. *hstore*. PostgreSQL 15 Documentation. 2023. URL: https://www.postgresql.org/docs/15/hstore.html (visited on 06/20/2023).

[16] Federico Di Gregorio. *Basic module usage*. Psycopg 2.9.6 documentation. 2021. URL: https://www.psycopg.org/docs/usage.html (visited on 06/19/2023).

[17] Ari Hakahuhta. "Turvallisuuskomitean pääsihteeri: Kun naapuri käy sotaa, kuntien ja yritysten olisi syytä rajoittaa karttojensa jakamista". In: *yle.fi* (May 2023). URL: https://yle.fi/a/74-20033447 (visited on 05/26/2023).

[18] Sarah Hoffmann. *Welcome to Pyosmium's documentation!* osmcode.org. 2020. URL: https://docs.osmcode.org/pyosmium/latest/ (visited on 06/07/2023).

[19] A. Le Guilcher, A.-M. Olteanu-Raimond, and M. B. Balde. "ANALYSIS OF MASSIVE IMPORTS OF OPEN DATA IN OPENSTREETMAP DATABASE: A STUDY CASE FOR FRANCE". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-4-2022 (2022), pp. 99–106. DOI: 10.5194/isprs-annals-V-4-2022-99-2022. URL: https://isprs-annals.copernicus.org/articles/V-4-2022/99/2022/ (visited on 05/22/2023).

[20] Michael Minn. *MMQGIS*. 2021. URL: http://michaelminn.com/linux/mmqgis (visited on 07/26/2023).

[21] osm2pgsql.org. *Osm2pgsql Manual*. 2023. URL: https://osm2pgsql.org/doc/manual.html (visited on 06/14/2023).

[22] osmcode.org. *Osmium Library*. URL: https://osmcode.org/libosmium/ (visited on 06/02/2023).

[23] osmcode.org. *Osmium manual pages*. URL: https://docs.osmcode.org/osmium/latest/ (visited on 06/02/2023).

[24] osmcode.org. *Osmium Tool Manual*. URL: https://osmcode.org/osmium-tool/manual.html (visited on 06/02/2023).

[25] python.org. *xml.etree.ElementTree — The ElementTree XML API*. Python Documentation. 2023. URL: https://docs.python.org/3/library/xml.etree.elementtree.html#parsing-xml (visited on 06/19/2023).

[26] A. Sarretta, M. Napolitano, and M. Minghini. "OPENSTREETMAP AS AN INPUT SOURCE FOR PRODUCING GOVERNMENTAL DATASETS: THE CASE OF THE ITALIAN MILITARY GEOGRAPHIC INSTITUTE". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLVIII-4/W7-2023 (2023), pp. 193–200. DOI: 10.5194/isprs-archives-XLVIII-4-W7-2023-193-2023. URL: https://isprs-archives.copernicus.org/articles/XLVIII-4-W7-2023/193/2023/ (visited on 07/25/2023).

List of Figures

List of Tables

Annexes

Source tags fetching with Python

Appendix A

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  6 14:48:43 2023

@author: alexysren
"""


### Imports

import os # to get our current directory path
import time # to compute the elapsed time for each function to return something

from tqdm import tqdm # make loops show a smart progress meter

from osmcha import changeset # OpenStreetMap Changeset Analyzer
import osmium # PyOsmium

import numpy as np # numpy
import pandas as pd # pandas library



### Global variables

current_directory_path = os.getcwd()



### Functions


class DataHandler(osmium.SimpleHandler):
    def __init__(self):
        osmium.SimpleHandler.__init__(self)
        self.elements = []

    def node(self, n):
        self.elements.append(["node", n.id, n.changeset])

    def way(self, w):
        self.elements.append(["way", w.id, w.changeset])

    def relation(self, r):
        self.elements.append(["relation", r.id, r.changeset])



def get_data(filepath):
    """
    imports data from an OSM data file and returns features id along its
    associated changeset id in a Panda Dataframe

    Parameter:
        filepath (str): path to OSM data file

    Return:
        (DataFrame): Panda DataFrame containing features id and associated changeset id
    """
    dataHandler = DataHandler()
    dataHandler.apply_file(filepath)
    col_names = ["type", "id", "changesetId"]
    return pd.DataFrame(dataHandler.elements, columns=col_names)



def get_source(changeset_id):
    """
    retrieves the source key value from an input changeset id

    Parameter:
        changeset_id (int): changeset id

    Return:
        (str): source key value
    """
    ch = changeset.Analyse(changeset_id)
    return ch.source

```

Figure A.1: Source tags fetching Homemade Python code (1/2)

```python


def apply_source(data, out_filename="data_with_source.csv"):
    """
    add the source key values column to the input that corresponds to each feature in
    the input data
    the result is also saved in a .csv file in root

    Parameter:
        data (DataFrame): Panda DataFrame containing features id and associated changeset id
        out_filename (str): filename for the output file (with its extension!)

    Return:
        (DataFrame): Panda DataFrame with columns [type, id, changesetId, source] (in order)
    """

    print("\n\n\n---------- Applying source process started... ---------- \n\n")
    start_time = time.time()


    # new_data = data[:50].copy()
    new_data = data.copy()
    source_column = np.array([])


    for i in tqdm(new_data.index):

        changeset_id = int(new_data.loc[i, "changesetId"])
        source = get_source(changeset_id)

        if source != "Not reported":
            source_column = np.append(source_column, source)
        else:
            source_column = np.append(source_column, "")

    new_data['source'] = source_column.tolist()



    new_data.to_csv(out_filename, float_format="%.3f", index_label='index', sep=" ")


    end_time = time.time()

    print("\n\n\n---------- Job's finished! ----------\nElapsed time: ", round(end_time - start_time, 2), "seconds\n\n\n")

    return new_data




### Main


if __name__ == '__main__':

    # Data imports

    roads_filename = "roads-uusimaa.osm.pbf"
    buidlings_filename = "buildings-uusimaa.osm.pbf"

    roads_uusimaa = get_data(current_directory_path + "/" + roads_filename)

    # Source values retrieval

    roads_uusimaa_source = apply_source(roads_uusimaa, "roads-uusimaa_source.csv")
```

Figure A.2: Source tags fetching Homemade Python code (2/2)

```
index type id changesetId source
0 node 130190 9486041
1 node 131938 6975707
2 node 131939 1188066
3 node 131940 2827488
4 node 131941 10996384
5 node 131954 11897653
6 node 131955 11897653
7 node 131956 11029829
8 node 131957 1370558
9 node 131979 3080664
10 node 131980 59214754 "Helsinki ortoilmakuva 2017"
11 node 131981 59214754 "Helsinki ortoilmakuva 2017"
12 node 131982 11614777
13 node 131983 11614777
14 node 131984 11614777
15 node 132101 134857011 "Survey and Helsinki region orthophoto"
16 node 132102 22946644
17 node 132103 62265520 "Helsingin ortokuva 2017"
18 node 132104 62265520 "Helsingin ortokuva 2017"
19 node 132105 62265520 "Helsingin ortokuva 2017"
20 node 132106 62265520 "Helsingin ortokuva 2017"
21 node 132107 17568343
22 node 132108 59214754 "Helsinki ortoilmakuva 2017"
23 node 132109 59351200 "Helsinki ortoilmakuva 2017, local_knowledge"
24 node 132110 428289
25 node 132111 5732909
26 node 132112 3778762
27 node 132113 118535241 survey
28 node 132116 59214754 "Helsinki ortoilmakuva 2017"
29 node 132118 6474559
30 node 134755 804632
31 node 134757 124682564 "HSY Orto;Mapillary;survey"
32 node 134759 57701713 "Vantaa ortoilmakuva 2017"
33 node 134766 645700
34 node 134767 124681122 "HSY Orto;Mapillary;survey"
35 node 134768 57701713 "Vantaa ortoilmakuva 2017"
36 node 134769 57701713 "Vantaa ortoilmakuva 2017"
37 node 134771 99992590 "local knowledge, aerial"
38 node 134772 6600395
39 node 134773 12732517
40 node 134774 6771897
41 node 134775 3196631
42 node 134776 5957783
43 node 134777 67983300
44 node 134778 67983300
45 node 134779 40464087
46 node 134783 16173294
47 node 134789 75374370
48 node 134790 12612790
49 node 134791 12612790
```

Figure A.3: Source tags fetching Homemade Python code — Preliminary results for the first 50 features from `roads-uusimaa.osm.pbf`

GDAL Configuration file for OSM import

Appendix B

Listing B.1: osmconf.ini (original version)

```
#
# Configuration file for OSM import
#

# put here the name of keys, or key=value, for ways that are assumed to be polygons if they are closed
# see http://wiki.openstreetmap.org/wiki/Map_Features
closed_ways_are_polygons=aeroway,amenity,boundary,building,craft,geological,historic,landuse,leisure,military,natural,office,place,shop,sport,tourism,highway=platform,public_transport=platform

# Uncomment to avoid laundering of keys ( ':' turned into '_' )
#attribute_name_laundering=no

# Some tags, set on ways and when building multipolygons, multilinestrings or other_relations,
# are normally filtered out early, independent of the 'ignore' configuration below.
# Uncomment to disable early filtering. The 'ignore' lines below remain active.
#report_all_tags=yes

# uncomment to report all nodes, including the ones without any (significant) tag
#report_all_nodes=yes

# uncomment to report all ways, including the ones without any (significant) tag
#report_all_ways=yes

[points]
# common attributes
osm_id=yes
osm_version=no
osm_timestamp=no
osm_uid=no
osm_user=no
osm_changeset=no

```

```
32 # keys to report as OGR fields
33 attributes=name,barrier,highway,ref,address,is_in,place,man_made
34 # keys that, alone, are not significant enough to report a node
      as a OGR point
35 unsignificant=created_by,converted_by,source,time,ele,attribution
36 # keys that should NOT be reported in the "other_tags" field
37 ignore=created_by,converted_by,source,time,ele,note,todo,
      openGeoDB:,fixme,FIXME
38 # uncomment to avoid creation of "other_tags" field
39 #other_tags=no
40 # uncomment to create "all_tags" field. "all_tags" and "
      other_tags" are exclusive
41 #all_tags=yes
42
43 [lines]
44 # common attributes
45 osm_id=yes
46 osm_version=no
47 osm_timestamp=no
48 osm_uid=no
49 osm_user=no
50 osm_changeset=no
51
52 # keys to report as OGR fields
53 attributes=name,highway,waterway,aerialway,barrier,man_made
54
55 # type of attribute 'foo' can be changed with something like
56 #foo_type=Integer/Real/String/DateTime
57
58 # keys that should NOT be reported in the "other_tags" field
59 ignore=created_by,converted_by,source,time,ele,note,todo,
      openGeoDB:,fixme,FIXME
60 # uncomment to avoid creation of "other_tags" field
61 #other_tags=no
62 # uncomment to create "all_tags" field. "all_tags" and "
      other_tags" are exclusive
63 #all_tags=yes
64
65 #computed_attributes must appear before the keywords _type and
      _sql
66 computed_attributes=z_order
67 z_order_type=Integer
68 # Formula based on https://github.com/openstreetmap/osm2pgsql/
      blob/master/style.lua#L13
69 # [foo] is substituted by value of tag foo. When substitution is
      not wished, the [ character can be escaped with \[ in literals
70 # Note for GDAL developers: if we change the below formula, make
      sure to edit ogrosmlayer.cpp since it has a hardcoded
      optimization for this very precise formula
71 z_order_sql="SELECT (CASE [highway] WHEN 'minor' THEN 3 WHEN '
      road' THEN 3 WHEN 'unclassified' THEN 3 WHEN 'residential' THEN
```

```
     3 WHEN 'tertiary_link' THEN 4 WHEN 'tertiary' THEN 4 WHEN '
    secondary_link' THEN 6 WHEN 'secondary' THEN 6 WHEN '
    primary_link' THEN 7 WHEN 'primary' THEN 7 WHEN 'trunk_link'
    THEN 8 WHEN 'trunk' THEN 8 WHEN 'motorway_link' THEN 9 WHEN '
    motorway' THEN 9 ELSE 0 END) + (CASE WHEN [bridge] IN ('yes', '
    true', '1') THEN 10 ELSE 0 END) + (CASE WHEN [tunnel] IN ('yes
    ', 'true', '1') THEN -10 ELSE 0 END) + (CASE WHEN [railway] IS
    NOT NULL THEN 5 ELSE 0 END) + (CASE WHEN [layer] IS NOT NULL
    THEN 10 * CAST([layer] AS INTEGER) ELSE 0 END)"
72
73 [multipolygons]
74 # common attributes
75 # note: for multipolygons, osm_id=yes instantiates a osm_id field
       for the id of relations
76 # and a osm_way_id field for the id of closed ways. Both fields
    are exclusively set.
77 osm_id=yes
78 osm_version=no
79 osm_timestamp=no
80 osm_uid=no
81 osm_user=no
82 osm_changeset=no
83
84 # keys to report as OGR fields
85 attributes=name,type,aeroway,amenity,admin_level,barrier,boundary
    ,building,craft,geological,historic,land_area,landuse,leisure,
    man_made,military,natural,office,place,shop,sport,tourism
86 # keys that should NOT be reported in the "other_tags" field
87 ignore=area,created_by,converted_by,source,time,ele,note,todo,
    openGeoDB:,fixme,FIXME
88 # uncomment to avoid creation of "other_tags" field
89 #other_tags=no
90 # uncomment to create "all_tags" field. "all_tags" and "
    other_tags" are exclusive
91 #all_tags=yes
92
93 [multilinestrings]
94 # common attributes
95 osm_id=yes
96 osm_version=no
97 osm_timestamp=no
98 osm_uid=no
99 osm_user=no
100 osm_changeset=no
101
102 # keys to report as OGR fields
103 attributes=name,type
104 # keys that should NOT be reported in the "other_tags" field
105 ignore=area,created_by,converted_by,source,time,ele,note,todo,
    openGeoDB:,fixme,FIXME
106 # uncomment to avoid creation of "other_tags" field
```

```
#other_tags=no
# uncomment to create "all_tags" field. "all_tags" and "other_tags" are exclusive
#all_tags=yes

[other_relations]
# common attributes
osm_id=yes
osm_version=no
osm_timestamp=no
osm_uid=no
osm_user=no
osm_changeset=no

# keys to report as OGR fields
attributes=name,type
# keys that should NOT be reported in the "other_tags" field
ignore=area,created_by,converted_by,source,time,ele,note,todo,openGeoDB:,fixme,FIXME
# uncomment to avoid creation of "other_tags" field
#other_tags=no
# uncomment to create "all_tags" field. "all_tags" and "other_tags" are exclusive
#all_tags=yes
```

First UML representation for uusimaa database

Appendix C

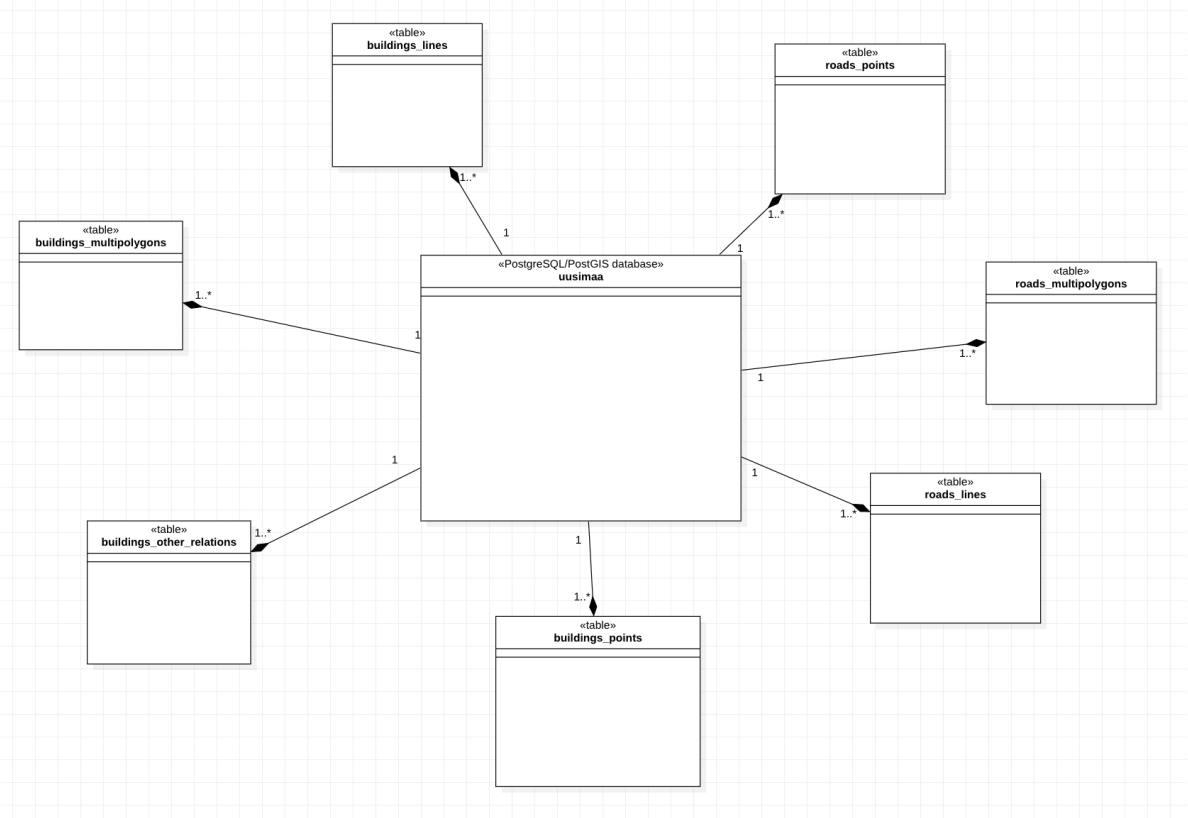


Figure C.1: `uusimaa` database UML Class diagram

Python script for importing changesets into uusimaa database

Appendix D

Listing D.1: chgset_to_pgsql.py Homemade Python script

```
1
2  #!/usr/bin/env python3
3  # -*- coding: utf-8 -*-
4  """
5  Created on Tue Jun 20 14:31:54 2023
6
7  @author: alexysren
8  """
9
10
11 """
12 ------------------------------------------------------------------------------

13
14                       OSM CHANGESET FILE IMPORT TO POSTGRESQL
15
16                     * custom tool by Alexys Ren - June 2023 *
17
18
19 ------------------------------------------------------------------------------

20 """
21
22
23 """
24 Important prerequisite: if it happens that you have the raw .osm.
      bz2 file, please extract the .osm from it
25              before launching this code
26 """
27
28
29
30 ### Librairies
31
32 from tqdm import tqdm
33
34 import xml.etree.ElementTree as ET
35
```

```
36 import psycopg2
37
38
39
40
41
42
43 ### Changeset file import
44
45 """ /!\ PLEASE MAKE SURE THIS CURRENT PYTHON FILE IS LOCATED IN
       THE SAME DIRECTORY AS YOUR
46                          CHANGESET FILE /!\
47 """
48 tree = ET.parse('uusimaa-changesets.osm') # feel free to change
       the filename here to fit your needs
49
50
51
52
53
54 ### Connecting to the PostgreSQL Database
55 # feel free again to change the connection details to fit your
       needs
56
57 conn = psycopg2.connect(database="uusimaa", user="postgres",
       password="postgres", host="localhost", port="5432")
58 cur = conn.cursor() # to perform database operations
59
60
61
62
63
64 ### Code
65
66 root = tree.getroot()
67
68
69 changesets = []
70
71 print("\n\n Parsing changesets... ")
72
73 for child in tqdm(root):
74     # every try/except below are meant to test if the related
          info exists!
75     try:
76         changeset_id = child.attrib['id']
77     except KeyError:
78         changeset_id = None
79     try:
80         created_at = child.attrib['created_at']
81     except KeyError:
```

```
82          created_at = None
83      try:
84          closed_at = child.attrib['closed_at']
85      except KeyError:
86          closed_at = None
87      try:
88          Open = child.attrib['open']
89      except KeyError:
90          Open = None
91      try:
92          user = child.attrib['user']
93      except KeyError:
94          user = None
95      try:
96          uid = child.attrib['uid']
97      except KeyError:
98          uid = None
99      try:
100         min_lat = child.attrib['min_lat']
101     except KeyError:
102         min_lat = None
103     try:
104         min_lon = child.attrib['min_lon']
105     except KeyError:
106         min_lon = None
107     try:
108         max_lat = child.attrib['max_lat']
109     except KeyError:
110         max_lat = None
111     try:
112         max_lon = child.attrib['max_lon']
113     except KeyError:
114         max_lon = None
115     try:
116         num_changes = child.attrib['num_changes']
117     except KeyError:
118         num_changes = None
119     try:
120         comments_count = child.attrib['comments_count']
121     except KeyError:
122         comments_count = None
123
124
125
126     # Preparing the additional tags for storing in hstore format
        (if they exist!)
127
128     additional_tags = []
129
130     try:
131         temp = child[0]
```

```
132     except IndexError:
133         additional_tags = None
134     else:
135         for tag in child:
136             key = tag.attrib['k']
137             value = tag.attrib['v']
138             additional_tags.append([key,value])
139 
140     changesets.append((changeset_id,created_at,closed_at,Open,
            user,uid,min_lat,min_lon,max_lat,max_lon,num_changes,
            comments_count,additional_tags))
141 
142 
143 
144 ### Inserting the output data into PostgreSQL
145 
146 # the targeted table structure has to match with the columns
        pattern below, and also the table name must be the same!
147 # feel free to change if needed!
148 
149 print("\n\n Inserting changesets into database... ")
150 
151 for changeset in tqdm(changesets):
152     cur.execute("""INSERT INTO changesets VALUES (%s, %s, %s, %s,
            %s, %s, %s, %s, %s, %s, %s, %s, hstore(%s));""", changeset
            )
153 
154 conn.commit() # Make the changes to the database persistent
155 
156 # Close communication with the database
157 cur.close()
158 conn.close()
```

NLS-related data proportions with additional 'created_by' and 'other_tags' tags

Appendix E

Tagging / Type of features	Points (49,353 features)	Lines (85 features)	Multipolygons (502,786 features)	All features (total)
NLS	0.008% *(4 features)*	0% *(0 features)*	6.6% *(33,159 features)*	6 %
Other	99.992% *(49,349 features)*	100% *(85 features)*	93.4% *(469,627 features)*	94 %

Table 3 — NLS-related features proportions for the "buildings" dataset (552,224 features)

Tagging / Type of features	Points (105,452 features)	Lines (468,982 features)	Multipolygons (4,214 features)	All features (total)
NLS	0.2% *(201 features)*	0.7% *(3,458 features)*	0.2% *(9 features)*	0.6%
Other	99.8% *(105,251 features)*	99.3% *(465,524 features)*	99.8% *(4,205 features)*	99.4%

Table 4 — NLS-related features proportions for the "roads" dataset (578,648 features)

Figure E.1: NLS-related data proportions with additional 'created_by' and 'other_tags' tags

SQL queries for categorised feature table creation

Appendix F

Those are examples for the `lines` layer from *roads* data set.

```sql
-- Making data output permanent (for QGIS visualisation)
CREATE TABLE nls_roads_lines AS

-- SELECT operation on features coming from the NLS (see WHERE condition below)
SELECT * FROM

-- LEFT JOIN table between roads_lines & changesets
(SELECT r.osm_id AS osm_feature_id, c.changeset_id,
r.source AS feature_source,
c.other_tags -> 'source' AS changeset_source,
r.wkb_geometry

FROM roads_lines AS r LEFT JOIN changesets AS c
ON r.osm_changeset = c.changeset_id) AS temp

-- one feature is selected if NLS is mentioned either in feature_source
-- or in changeset_source
WHERE
temp.feature_source ILIKE '%mml%' OR
temp.feature_source ILIKE '%nls%' OR
temp.feature_source ILIKE '%maanmittauslaitos%' OR
temp.feature_source ILIKE '%national land survey of finland%' OR
temp.feature_source ILIKE '%finnish geospatial research institute%' OR
temp.feature_source ILIKE '%fgi%' OR
temp.feature_source ILIKE '%Paikkatietokeskus%' OR
temp.feature_source ILIKE '%lmv%' OR
temp.feature_source ILIKE '%Lantmäteriverket%' OR
temp.feature_source ILIKE '%Geodatacentralen%' OR

temp.changeset_source ILIKE '%mml%' OR
temp.changeset_source ILIKE '%nls%' OR
temp.changeset_source ILIKE '%maanmittauslaitos%' OR
temp.changeset_source ILIKE '%national land survey of finland%' OR
temp.changeset_source ILIKE '%finnish geospatial research institute%' OR
temp.changeset_source ILIKE '%fgi%' OR
temp.changeset_source ILIKE '%Paikkatietokeskus%' OR
temp.changeset_source ILIKE '%lmv%' OR
temp.changeset_source ILIKE '%Lantmäteriverket%' OR
temp.changeset_source ILIKE '%Geodatacentralen%';
```

Figure F.1: '*nls*' tables SQL creation code

```sql
-- Making data output permanent (for QGIS visualisation)
CREATE TABLE no_source_roads_lines AS

-- SELECT operation on features with no source (see WHERE condition below)
SELECT * FROM

-- LEFT JOIN table between roads_lines & changesets
(SELECT r.osm_id AS osm_feature_id, c.changeset_id,
r.source AS feature_source,
c.other_tags -> 'source' AS changeset_source,
r.wkb_geometry

FROM roads_lines AS r LEFT JOIN changesets AS c
ON r.osm_changeset = c.changeset_id) AS temp


WHERE temp.feature_source IS NULL
AND
temp.changeset_source IS NULL;
```

Figure F.2: '*no_source*' tables SQL creation code

```sql
-- Making data output permanent (for QGIS visualisation)
CREATE TABLE other_roads_lines AS

-- SELECT operation on features that are neither from the NLS and neither with no source (see WHERE condition below)
SELECT * FROM roads_lines

WHERE NOT EXISTS (
   SELECT 1
   FROM nls_roads_lines
   WHERE nls_roads_lines.osm_feature_id = roads_lines.osm_id
)
AND NOT EXISTS (
   SELECT 1
   FROM no_source_roads_lines
   WHERE no_source_roads_lines.osm_feature_id = roads_lines.osm_id
);
```

Figure F.3: '*other*' tables SQL creation code

SQL query to select the most prominent occurrences of NLS-like source values

Appendix G

```sql
SELECT * FROM (
    SELECT temp.source, SUM(temp.occurences) AS occurences FROM (
        SELECT feature_source AS source, COUNT(*) AS occurences FROM nls_buildings_multipolygons GROUP by source
        UNION
        SELECT changeset_source, COUNT(*) FROM nls_buildings_multipolygons GROUP BY changeset_source
    ) AS temp
    GROUP BY temp.source
) AS tmp

WHERE
tmp.source ILIKE '%mml%' OR
tmp.source ILIKE '%nls%' OR
tmp.source ILIKE '%maanmittauslaitos%' OR
tmp.source ILIKE '%national land survey of finland%' OR
tmp.source ILIKE '%finnish geospatial research institute%' OR
tmp.source ILIKE '%fgi%' OR
tmp.source ILIKE '%Paikkatietokeskus%' OR
tmp.source ILIKE '%lmv%' OR
tmp.source ILIKE '%Lantmäteriverket%' OR
tmp.source ILIKE '%Geodatacentralen%'

ORDER BY tmp.occurences DESC;
```

Figure G.1: SQL query to select the most prominent occurrences of NLS-like source values

Python script for OSM history file import into PostgreSQL

Appendix H

Listing H.1: `history_to_pgsql_enhanced.py` Homemade Python script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 19 11:42:19 2023

@author: alexysren
"""



"""
--------------------------------------------------------------------------------

                    OSM HISTORY FILE IMPORT INTO POSTGRESQL
                            [ENHANCED VERSION]

                   * custom tool by Alexys Ren     August 2023 *


--------------------------------------------------------------------------------

"""

"""
User guideline: Please take a look at the 'MAIN' part at the end of this script,
                so to make the necessary changes to import your own OSM history file !
"""




### Librairies import

from tqdm import tqdm    # for displaying the elapsed time after launching the code, and the number of iterations per second
```

```
34 import psycopg2    # PostgreSQL driver for Python support
35 from psycopg2 import sql # for generating dynamically SQL queries
       (for choosing dynamically a table name)
36 import lxml.etree    # for processing XML data, using C-based
       libraries libxml2 and libxslt     superior to the native
       ElementTree API in terms of processing time and functionalities
37
38
39
40
41 ### Code
42
43
44 def history_importer(osm_history_filename, pgsql_tablename='
       OSM_history'):
45     """
46     imports an OSM history file (XML) as a new table in the
           PostgreSQL database
47     provided by the user in the 'MAIN'
48
49     Parameters
50     ----------
51     osm_history_filename : string
52         OSM history filename with its extension.
53     pgsql_tablename : string
54         Tablename for output PostgreSQL table.
55
56     Returns
57     -------
58     None.
59
60     """
61
62     # Creating a new PostgreSQL table for hosting history data
63     cur.execute(sql.SQL("""CREATE TABLE public.{}
64 (
65     id bigint,
66     osm_id bigint,
67     osm_type text,
68     version integer,
69     "timestamp" timestamp without time zone,
70     uid bigint,
71     "user" text,
72     changeset_id bigint,
73     visible boolean,
74     lat double precision,
75     lon double precision,
76     members_refs bigint[],
77     other_tags hstore,
78     PRIMARY KEY (id)
79 );""").format(sql.Identifier(pgsql_tablename)))
```

```
80
81      # Making the previous change to the database persistent
82      conn.commit()
83
84
85      with open(osm_history_filename, "rb") as f:
86
87          id_primary_key = 0
88
89
90          print("\n\n Parsing history from XML, on-the-fly
                importing... ")
91          print("\n N.B.: For reference, importing a 7.73GB history
                 file took me 1:30h (around 22 million feature versions
                ). Please also consider your memory capacity!")
92          print("\n\n ** Ongoing process, please make sure the
                screen saver mode is disabled on your computer! (
                otherwise the kernel will be interrupted!) **")
93
94          for event, child in tqdm(lxml.etree.iterparse(f)):
95
96
97              osm_type = child.tag
98
99              if osm_type == 'osm':
100                 # Ignore 'OSM' XML tag (metadata)
101                 continue
102
103             if osm_type in ['node', 'way', 'relation']:
104
105                 try:
106                     if osm_id == child.attrib['id'] and version
                            == child.attrib['version']:
107                         """ The closing XML tag for a OSM feature
                                can either be '/>' or either like '</
                            feature_type>'.
108                         In the case it is '</feature_type>', we
                            jump directly to the next iteration, as
                             the iterparse function considers this
                            closing XML tag as a OSM feature in its
                             own right
109                         with all the information related to the
                            very previous feature """
110                         continue
111                 except UnboundLocalError: # triggered only once
                        at the first iteration since both 'osm_id' and
                        'version' are not initialised yet!
112                     pass
113
114                 osm_id = child.attrib['id']
115                 version = child.attrib['version']
```

```
116
117                 id_primary_key += 1 # inserting a new id column
                       so every record has a unique identifier
118                                     # (one feature can have
                                           several versions using the
                                           same feature ID!)
119                 # every try/except below is meant to test if the
                       related info exists!
120                 try:
121                     timestamp = child.attrib['timestamp']
122                 except KeyError:
123                     timestamp = None
124                 try:
125                     uid = child.attrib['uid']
126                 except KeyError:
127                     uid = None
128                 try:
129                     user = child.attrib['user']
130                 except KeyError:
131                     user = None
132                 try:
133                     changeset_id = child.attrib['changeset']
134                 except KeyError:
135                     changeset_id = None
136                 try:
137                     visible = child.attrib['visible']
138                 except KeyError:
139                     visible = None
140
141
142                 if osm_type == 'node': # if the OSM object is a
                       node
143                     try:
144                         lat = child.attrib['lat']
145                     except KeyError:
146                         lat = None
147                     try:
148                         lon = child.attrib['lon']
149                     except KeyError:
150                         lon = None
151                 else:
152                     lat = None
153                     lon = None
154
155
156
157                 # Saving currently available data on the current
                       OSM feature in a list
158                 history_record = [id_primary_key,
159                                   osm_id,
160                                   osm_type,
```

```
                                        version,
                                        timestamp,
                                        uid,
                                        user,
                                        changeset_id,
                                        visible,
                                        lat,
                                        lon]

                    """
                        (A): The 'etree.iterparse' method from lxml
                           read all tags and members related to any
                           OSM feature before the feature itself.
                           Below, we retrieve any additional
                           information
                        about the current OSM feature, and append it
                           to the end of 'history_record' list.
                    """

                    try:
                        if len(next_members_refs) > 0:
                            # If member references were read
                            history_record.append(next_members_refs)
                        else:
                            history_record.append(None)
                    except UnboundLocalError: # triggered only once
                       at the first iteration since 'next_members_refs
                       ' is not initialised yet!
                        history_record.append(None)


                    try:
                        if len(next_additional_tags) > 0:
                            # If tags were read
                            history_record.append(
                               next_additional_tags)
                        else:
                            history_record.append(None)
                    except UnboundLocalError: # triggered only once
                       at the first iteration since '
                       next_additional_tags' is not initialised yet!
                        history_record.append(None)


                    """
                        No more information about the current OSM
                           feature is contained in the XML file, we
                           proceed with the insertion phase.
                    """

                    # SQL query for inserting the data
```

```
200                 # N.B.: the 'INSERT' operation is repeated over
                        all the OSM feature versions in the OSM history
                        file
201                 cur.execute(sql.SQL("""INSERT INTO {} VALUES (%s,
                        %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
                        hstore(%s));""").format(sql.Identifier(
                        pgsql_tablename)), history_record)
202
203                 # Make the changes to the database persistent
204                 conn.commit()
205
206                 # Freeing the memory
207                 del history_record
208
209
210                 # Freeing memory and variable initialisation for
                        the next iteration
211                 try:
212                     del next_members_refs
213                 except NameError: # triggered only once at the
                        first iteration since 'next_members_refs' is
                        not initialised yet!
214                     pass
215                 next_members_refs = []
216                 try:
217                     del next_additional_tags
218                 except NameError: # triggered only once at the
                        first iteration since 'next_additional_tags' is
                        not initialised yet!
219                     pass
220                 next_additional_tags = []
221
222
223
224
225
226             else:
227                 # if the XML element is actually a OSM tag
228                 """
229                     See (A).
230                 """
231
232
233                 # if the tag is a referenced member of a way or a
                        relation
234                 if osm_type == "nd" or osm_type == "member":
235
236                     member_id = int(child.attrib['ref'])
237                     next_members_refs.append(member_id)
238
239
```

```
240                 # if the tag is just a common attribute
241                 else:
242
243                     key = child.attrib['k']
244                     value = child.attrib['v']
245
246                     tag = [key,value]
247
248                     next_additional_tags.append(tag)
249
250
251             # Freeing the memory for the next iteration
252             child.clear()
253
254     print("\n\n Process completed! \n Your OSM history file
        content should now be appearing in your '", pgsql_tablename
        , "' PostgreSQL table!")
255
256
257
258
259 ### Main     Code execution
260
261
262 if __name__ == '__main__':
263
264     # Connecting to the PostgreSQL Database
265     conn = psycopg2.connect(database="uusimaa", user="postgres",
        password="postgres", host="localhost", port="5432")
266
267     # Opening a cursor to perform database operations
268     cur = conn.cursor()
269
270     # Selecting the OSM history file for import
271     """
272     Important prerequisite: if it happens that you have the raw .
        osh.pbf file,
273                             please convert it to .osh format (XML
                                ) before launching this code
274                             ('osmium cat' command can be used for
                                 that purpose)
275     """
276     #filename = 'otaniemi-history.osh'
277     filename = 'uusimaa-history.osh'
278
279     """ /!\ PLEASE MAKE SURE THIS CURRENT PYTHON FILE IS LOCATED
        IN THE SAME DIRECTORY AS YOUR
280                             HISTORY FILE /!\
281     """
282
283     # Choosing a name for the output PostgreSQL table (optional,
```

```
        default='OSM_history')
    tablename = 'history'

    # Importing history file content
    history_importer(filename, tablename)


    # Close communication with the database
    cur.close()
    conn.close()
```

Naive Python script for OSM history file import into PostgreSQL

Appendix I

Listing I.1: `history_to_pgsql.py` Homemade Python script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 19 11:42:19 2023

@author: alexysren
"""



"""
-------------------------------------------------------------------------------

                     OSM HISTORY FILE IMPORT INTO POSTGRESQL

                    * custom tool by Alexys Ren     July 2023 *


-------------------------------------------------------------------------------
"""


"""
Important prerequisite: if it happens that you have the raw .osh.pbf file,
                        please convert it to .osh format (XML) before launching this code
                        (osmium cat command can be used for that purpose)
"""



### Librairies

from tqdm import tqdm
```

```
34
35 import xml.etree.ElementTree as ET
36
37 import psycopg2
38
39
40
41
42
43 ### History file import
45
46 """ /!\ PLEASE MAKE SURE THIS CURRENT PYTHON FILE IS LOCATED IN
       THE SAME DIRECTORY AS YOUR
47                          HISTORY FILE /!\
48 """
49 print("\n\n Parsing XML... ")
50 tree = ET.parse('otaniemi-history.osh') # feel free to change the
       filename here to fit your needs
51
52
53
54
55
56 ### Connecting to the PostgreSQL Database
57 # feel free again to change the connection details to fit your
       needs
58
59 conn = psycopg2.connect(database="uusimaa", user="postgres",
       password="postgres", host="localhost", port="5432")
60 cur = conn.cursor() # to perform database operations
61
62
63
64
65
66 ### Code
67
68 root = tree.getroot()
69
70
71 history_records = []
72 id_primary_key = 0
73
74
75 print("\n\n Parsing history... ")
76
77 for child in tqdm(root):
78     id_primary_key += 1 # inserting a new id column so every
          record has a unique identifier
79                         # (one feature can have several versions
```

```
                            using the same feature ID!)
    # every try/except below are meant to test if the related
       info exists!
    try:
        osm_id = child.attrib['id']
    except KeyError:
        osm_id = None
    osm_type = child.tag
    try:
        version = child.attrib['version']
    except KeyError:
        version = None
    try:
        timestamp = child.attrib['timestamp']
    except KeyError:
        timestamp = None
    try:
        uid = child.attrib['uid']
    except KeyError:
        uid = None
    try:
        user = child.attrib['user']
    except KeyError:
        user = None
    try:
        changeset_id = child.attrib['changeset']
    except KeyError:
        changeset_id = None
    try:
        visible = child.attrib['visible']
    except KeyError:
        visible = None


    if child.tag == 'node': # if the OSM object is a node
        try:
            lat = child.attrib['lat']
        except KeyError:
            lat = None
        try:
            lon = child.attrib['lon']
        except KeyError:
            lon = None
    else:
        lat = None
        lon = None
```

```
129
130
131     # Preparing the additional tags for storing in hstore format
            (if they exist!)
132     # and Saving the node references for the ways, and the member
            (node, way, or relation) references for the relations
133     #     --> In fact, geometry information is only contained in
            nodes. So to access the geometry
134     #         of a way, you need the geometries of the nodes that
            are part of this way. The same goes
135     #         for the geometries of relations.
136
137     additional_tags = None
138     members_refs = None # references to nodes for ways and to
            members for relations
139
140     # For initialising the storage list (see below)
141     stop1 = True
142     stop2 = True
143
144     # Checking if the OSM object has additional tags
145     try:
146         temp = child[0]
147     except IndexError:
148         pass
149     else:
150         for tag in child:
151
152             # if the tag is a referenced member of a way or a
                    relation
153             if (child.tag == "way" and tag.tag == "nd") or (child
                    .tag == "relation" and tag.tag == "member"):
154
155                 if stop1: # if it's the first referenced member
                        encountered, initialise an empty list for
                        storing
156                     members_refs = []
157                     stop1 = False
158                     # NB: this initialisation won't happen again,
                            since 'stop1' variable is not used anymore
                            from now on
159
160                 members_refs.append(int(tag.attrib['ref'])) #
                        adding the member id to the list
161
162             # if the tag is just a common attribute
163             else:
164
165                 if stop2: # if it's the first additional tag
                        encountered, initialise an empty list for
                        storing
```

```
                        additional_tags = []
                        stop2 = False
                        # NB: same remarks as for 'stop1' variable

                    key = tag.attrib['k']
                    value = tag.attrib['v']
                    additional_tags.append([key,value])

        history_records.append((id_primary_key,
                                osm_id,
                                osm_type,
                                version,
                                timestamp,
                                uid,
                                user,
                                changeset_id,
                                visible,
                                lat,
                                lon,
                                members_refs,
                                additional_tags))


### Inserting the output data into PostgreSQL

# the targeted table structure has to match with the columns pattern below, and also the table name must be the same!
# feel free to change if needed!

print("\n\n Inserting history into database... ")

for history_record in tqdm(history_records):
    cur.execute("""INSERT INTO otaniemi_history VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, hstore(%s));""", history_record)

conn.commit() # Make the changes to the database persistent

# Close communication with the database
cur.close()
conn.close()
```

OSM Feature-level modifications analyser Python script

Appendix J

Listing J.1: typology_modif_encoding_copy.py Homemade Python script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 20 17:01:22 2023

@author: alexysren
"""

"""
------------------------------------------------------------------------------


                    OSM FEATURE-LEVEL MODIFICATIONS ANALYSER
                                [BETA VERSION]

                   * custom tool by Alexys Ren     July 2023 *


------------------------------------------------------------------------------

"""

"""
['BETA VERSION']: Since some challenges were encountered in our investigation, the initial goal was
rerouted toward statistics on geometry modification. However, all the functions below were implemented so to be easily developed further to achieve the initial goal.

For more information, please see my 'internship_report'!
"""

"""
[Quick start]: If you would like to use the code as it is, just modify the 'MAIN' part at the end of the code to fit your needs !
"""

```

```
32
33 ### Libraries import
34
35 from tqdm import tqdm   # for displaying a progress bar on loops
36 import os
37
38 import psycopg2     # PostgreSQL driver for Python support
39 import pandas as pd
40 import numpy as np
41
42
43
44 ### Functions
45
46
47 def osm_versions(osm_id):
48     """
49     Returns all the versions of a OSM feature based on its id
50
51     Parameters
52     ----------
53     osm_id : int
54         OSM FEATURE ID.
55
56     Returns
57     -------
58     pandas DataFrame
59         SQL QUERY RESULT.
60
61     """
62     cur.execute("SELECT * FROM history WHERE osm_id = %s ORDER BY
           version;", (osm_id,))
63     return pd.DataFrame(cur.fetchall(), columns=['id',
64                                                  'osm_id',
65                                                  'osm_type',
66                                                  'version',
67                                                  'timestamp',
68                                                  'uid',
69                                                  'user',
70                                                  'changeset_id',
71                                                  'visible',
72                                                  'lat',
73                                                  'lon',
74                                                  'members_refs',
75                                                  'other_tags'])
76
77
78
79
80 def geometry_references_resolver(member_versions,
      feature_timestamp):
```

```
81      """
82      When resolving references for accessing ways and relations
           geometries,
83      referenced members may also have several versions to be taken
            into account.
84
85      This function takes all the versions of a referenced member
           from a way or a relation,
86      and return a unique version from it.
87
88      This unique version of the referenced member corresponds to
89      its latest version while being older than the feature's
           timestamp.
90
91      Parameters
92      ----------
93      member_versions : pandas DataFrame
94          CONTAINS ALL THE VERSIONS OF A REFERENCED MEMBER.
95      feature_timestamp : pandas Timestamp
96          TIMESTAMP FOR THE OSM FEATURE TO WHICH THE REFERENCE IS
               LINKED.
97
98      Returns
99      -------
100     pandas Series
101         UNIQUE REFERENCED MEMBER FOR THE WAY/RELATION.
102
103     """
104
105     # Calculating time difference
106     member_versions['time_difference'] = member_versions['
           timestamp'] - feature_timestamp
107
108     # Filtering versions older than 'feature_timestamp'
109     filtered_df = member_versions[member_versions['timestamp'] <=
            feature_timestamp]
110
111     n, m = filtered_df.shape
112     if n == 0:
113         # tests revealed that some referenced members can
114         # be younger (in time) than the ways or relations they
               are referenced to, which
115         # is theorically impossible: DATA error (Error handling
               in 'geometry_modification')
116         return filtered_df
117
118     # Find the version with minimal time difference from the
           filtered DataFrame
119     min_time_difference_index = filtered_df['time_difference'].
           idxmax() # relative time!
120     return member_versions.iloc[min_time_difference_index]
```

```
121
122
123
124
125
126
127 ## Topology of modifications
128
129 # Geometry modification
130
131 def geometry_modification(t1,t2,osm_type):
132     """
133     [RECURSIVE FUNCTION]
134     returns 1 if there has been a geometry modification, None if
        not,
135     or 2 if there has been an error in the process
136
137     Parameters
138     ----------
139     t1 : pandas Series
140         PREVIOUS VERSION.
141     t2 : pandas Series
142         CURRENT VERSION.
143     osm_type : string
144         FEATURE TYPE.
145
146     Returns
147     -------
148     int
149         see main description.
150
151     """
152
153     if osm_type == 'node':
154         if (t1.lat == t2.lat and t1.lon == t2.lon) or (np.isnan(
            t2.lat) and np.isnan(t2.lon)):
155             # if coordinates are equals or if the geometry has
                been erased
156             return None
157         return 1
158
159     # So frow now on, only ways and relations are considered
160     t1_members_refs = t1['members_refs']
161     t2_members_refs = t2['members_refs']
162
163     if t2_members_refs == None:
164         # if the way/relation geometry has been erased
165         return None
166
167     if t1_members_refs == None:
168         # if the way/relation has no referenced members: DATA
```

```
            error
169         #   -> it's theorically impossible since way and relation
                existences relies
170         #       on their references!
171         return 2
172
173     n, m = len(t1_members_refs), len(t2_members_refs)
174
175     if n != m:
176         # if the numbers of nodes/members are not equal, there
                has been a modification!
177         return 1
178
179     t1_timestamp = t1['timestamp']
180     t2_timestamp = t2['timestamp']
181
182
183     if osm_type == 'way':
184         # Iterating over referenced nodes
185         for i in range(n):
186
187             # Referenced nodes from ways may also have several
                    versions needed to be taken into account
188             t1_i_node_versions = osm_versions(t1_members_refs[i])
189             t2_i_node_versions = osm_versions(t2_members_refs[i])
190
191             # If the referenced node doesn't exist in our
                    database
192             #   -> cut during the geographical extraction of
                    Otaniemi: EDGE EFFECT error
193             if t1_i_node_versions.shape[0] == 0 or
                    t2_i_node_versions.shape[0] == 0:
194                 return 2
195
196
197             # Selecting the right version of the referenced node
                    based on timestamps
198             #print("t=", t1_i_node_versions['timestamp'], "
                    timestamp=",t1_timestamp)
199             t1_i_node = geometry_references_resolver(
                    t1_i_node_versions, t1_timestamp)
200             t2_i_node = geometry_references_resolver(
                    t2_i_node_versions, t2_timestamp)
201
202             if t1_i_node.shape[0] == 0 or t2_i_node.shape[0] ==
                    0:
203                 # if there has been an error (see '
                        geometry_references_resolver' inline comments)
204                 return 2
205
206             # Comparing nodes geometry
```

```
207             if geometry_modification(t1_i_node,t2_i_node,'node')
                    != None:
208                 return 1
209         return None
210
211     else: # if osm_type == 'relation':
212         # Iterating over referenced nodes/ways/relations
213         for i in range(n):
214
215             # Referenced members from relations may also have
                    several versions needed to be taken into account
216             t1_i_member_versions = osm_versions(t1_members_refs[i
                    ])
217             t2_i_member_versions = osm_versions(t2_members_refs[i
                    ])
218
219             # If the referenced member doesn't exist in our
                    database
220             #   -> cut during the geographical extraction of
                    Otaniemi: EDGE EFFECT error
221             if t1_i_member_versions.shape[0] == 0 or
                    t2_i_member_versions.shape[0] == 0:
222                 return 2
223
224             # Selecting the right version of the referenced
                    member based on timestamps
225             t1_i_member = geometry_references_resolver(
                    t1_i_member_versions, t1_timestamp)
226             t2_i_member = geometry_references_resolver(
                    t2_i_member_versions, t2_timestamp)
227
228
229             if t1_i_member.shape[0] == 0 or t2_i_member.shape[0]
                    == 0:
230                 # if there has been an error (see '
                        geometry_references_resolver' inline comments)
231                 return 2
232
233
234             # Comparing ways or relations geometry
235
236             t1_i_member_type = t1_i_member['osm_type']
237             t2_i_member_type = t2_i_member['osm_type']
238
239                 # if the feature types are different between the
                        compared members, there has been a modification
                        !
240             if t1_i_member_type != t2_i_member_type:
241                 return 1
242
243             if geometry_modification(t1_i_member, t2_i_member,
```

```
                t1_i_member_type) != None:
244                  return 1
245         return None
246
247
248 # Semantic modifications (not our case study, to be filled if
       needed!)
249
250 def tag_enrichment(t1,t2):
251     # same etc.
252     return None
253
254 def tag_suppression(t1,t2):
255     return None
256
257 def tag_modification(t1,t2):
258     return None
259
260
261
262 ## Encoding modification types into sequence
263 """
264
265 [IMPORTANT NOTE]: The below function was originally meant for
       preparing the clustering analysis
266 based on encoded sequences of modifications with semantic ones
       included. It is now used
267 for geometry-focused studies, but can be developed further on to
       fit your needs.
268
269 """
270
271 def sequence_modifications(osm_id):
272     """
273     returns the encoded sequence of modifications for one OSM
          feature based on its id
274
275     Parameters
276     ----------
277     osm_id : int
278         OSM FEATURE ID.
279
280     Returns
281     -------
282     seq : int list
283         SEQUENCE OF MODIFICATIONS.
284
285     """
286
287     # Retrieving all the feature versions
288     versions = osm_versions(osm_id)
```

```
289     n, m = versions.shape # n = number of versions, m = number of
            fields
290
291     if n < 1:
292         # tests has shown that it may happen a OSM feature is not
                recorded in the history
293         return [2]
294
295     seq = [] # sequence of modifications
296
297
298     # Retrieving the feature type (is it a node, a way, or a
            relation?)
299     feature_type = versions['osm_type'][0]
300
301
302     # Iterating over versions (Pairwise similarity comparing)
303     for i in range(n-1):
304         geometryModification = geometry_modification(versions.
                iloc[i], versions.iloc[i+1], feature_type)
305         if geometryModification != None:
306             seq.append(geometryModification)
307
308         # tag_enrichment =
309         # if tag_enrichment is not None:
310             # seq.append(tag_enrichment(versions.iloc[i],versions
                    .iloc[i+1]))
311
312         # tag_suppression =
313         # tag_modification =
314
315         # .
316         # .
317         # . and so on...
318
319     return seq
320
321
322
323
324 def nb_geometry_modification(osm_id):
325     """
326     based on the current state of the 'sequence_modifications'
            function,
327     nb_geometry_modification returns the number of geometry
            modifications
328     underwent by an input OSM feature based on its ID.
329
330     Parameters
331     ----------
332     osm_id : int
```

```
333         OSM FEATURE ID.
334
335     Returns
336     -------
337     int
338         NUMBER OF GEOMETRY MODIFICATIONS FOR THE INPUT FEATURE.
339
340     """
341     seq = sequence_modifications(osm_id)
342     if 2 in seq:
343         return None
344     return len(seq)
345
346
347
348 def get_geometry_modification():
349     """
350     saves locally and returns a pandas DataFrame containing one
          line per feature with its
351     associated number of geometry modifications.
352
353     Returns
354     -------
355     df : pandas DataFrame
356         see main description.
357
358     """
359     # Retrieving OSM feature ids from '
          nls_buildings_multipolygons'
360     cur.execute("""SELECT * FROM
361
362 (SELECT CAST(osm_id AS bigint) FROM nls_buildings_multipolygons
363 UNION
364 SELECT CAST(osm_way_id AS bigint) FROM
       nls_buildings_multipolygons) AS temp
365
366 WHERE temp.osm_id IS NOT NULL;""")
367
368     df = pd.DataFrame(cur.fetchall(), columns=['osm_id'])
369
370     nbGeometryModification = []
371
372     for osm_id in tqdm(df.osm_id):
373         nbGeometryModification.append(nb_geometry_modification(
              osm_id))
374
375     df['nb_geometry_modification'] = nbGeometryModification
376
377     df.to_csv("
          nls_buildings_multipolygons_nb_geometry_modification.csv",
          float_format="%.3f", index_label="index", sep=" ")
```

```
    return df


## Typology of modifications: Statistics


def massive_contributions_extract(filename):
    # /!\ TO BE MODIFIED:
        # additional input: feature id list containing ids of NLS
            massive contributions
        # filter the below dataframe to return only features
           coming from NLS massive contributions
        # thus allowing proper statistics to answer our initial
           problem

    """ [IN REALITY]: this function is just about loading a .csv
       file, saved previously to avoid launching '
       get_geometry_modification' at each code running
                    (let's say you want to display the statistics
                        again later...).

                    Indeed, 'get_geometry_modification' is the
                       most time consuming in the whole process! (
                       the whole history is scanned!) """
    df = pd.read_csv(os.path.join(os.getcwd(), filename),
       delimiter=" ")
    return df



def modifications_analyser(dataframe):
    """
    displays detailed statistics about modifications underwent by
        the input features
    (can be developed further on to include semantic
       modifications)

    Parameters
    ----------
    dataframe : pandas DataFrame
        OSM FEATURES WITH NUMBER OF GEOMETRY MODIFICATIONS.

    Returns
    -------
    None.

    """
    nb_features = dataframe.shape[0]
    nb_geometry_modification = dataframe['
       nb_geometry_modification']
    arr = np.array(nb_geometry_modification)
```

```
417
418     nb_nan_values = nb_features - nb_geometry_modification.count
            ()
419
420     # Creating boolean masks for non-zero and non-'NaN' values
421     non_zero_mask = arr != 0
422     non_nan_mask = ~np.isnan(arr)
423     # Counting the number of values that satisfy both conditions
424     nb_non_zero_and_non_nan_values = np.count_nonzero(np.
            logical_and(non_zero_mask, non_nan_mask))
425
426     # Creating a boolean mask for zero values
427     zero_mask = arr == 0
428     # Counting the number of zero values using the boolean mask
429     nb_zero_values = np.count_nonzero(zero_mask)
430
431
432     print("\n\n----------------  Geometry modification statistics
            ----------------\n")
433     print("Feature count:", nb_features)
434     print("Among the ", nb_features, "features")
435     print("    -",nb_non_zero_and_non_nan_values,"underwent
            geometric modifications  (", round(100*
            nb_non_zero_and_non_nan_values/nb_features, 1), "% )")
436     print("    -",nb_zero_values,"have not been modified  (",
            round(100*nb_zero_values/nb_features, 1), "% )")
437     print("    -",nb_nan_values,"could not be analysed  (", round
            (100*nb_nan_values/nb_features, 1), "% )")
438     print("*    Average number of geometry modifications per
            feature:", round(nb_geometry_modification.mean(), 1))
439     print("*    Max number of geometry modifications:",
            nb_geometry_modification.max(), "\n")
440
441
442
443 ### Code execution     'MAIN'
444
445 if __name__ == '__main__':
446
447     # # Connecting to PostgreSQL database (feel free to change
            the details to fit your needs)
448     # conn = psycopg2.connect(database="uusimaa", user="postgres
            ", password="postgres", host="localhost", port="5432")
449
450     # # Opening a cursor to perform database operations
451     # cur = conn.cursor()
452
453
454     """ /!\ This code assumes the existence of a PostgreSQL table
            containing your OSM history data, which also need to
            respect a particular table structure.
```

```
455             If you haven't already done so, please import your
                   history data using the 'history_to_pgsql_enhanced.
                   py' Python script,
456             then ensure that the SQL query in the 'cur.execute'
                   method of 'osm_versions' function matches the name
                   of your history table!  /!\
457
458         N.B.: Modifying the SQL query in the code was a coding
                mistake,
459                as I should have asked the user for a tablename
                      input in this 'MAIN' directly to make this easier
460     """
461
462     # # Saving the number of geometry modifications for each
           feature belonging to 'nls_buildings_multipolygons' table on
            our computer
463     # get_geometry_modification()
464
465     # Importing the saved file (from 'get_geometry_modification')
466     nls_buildings_multipolygons = massive_contributions_extract("
           nls_buildings_multipolygons_nb_geometry_modification.csv")
467
468     # Displaying statistics on geometry modification
469     modifications_analyser(nls_buildings_multipolygons)
470
471     # # Closing communication with the database
472     # cur.close()
473     # conn.close()
```

Python script for retrieving OSM ids from feature which underwent geometric modifications

Appendix K

Listing K.1: `get_geom_modif_ids.py` Homemade Python script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Aug  8 10:52:14 2023

@author: alexysren
"""

### Libraries import

import os

import pandas as pd
import numpy as np


### Code


def get_geom_modif_ids(filename):
    """
    create a new .csv file containing the OSM feature ids of
       those that have
    underwent geometry modification from the input file

    Parameters
    ----------
    filename : string
        CSV filename with its extension.

    Returns
    -------
    None.

    """

```

```
36      df = pd.read_csv(os.path.join(os.getcwd(), filename),
            delimiter=" ")
37
38      osm_id = df['osm_id']
39      nb_geometry_modification = df['nb_geometry_modification']
40      arr = np.array(nb_geometry_modification)
41
42      # Creating boolean masks for non-zero and non-'NaN' values
43      non_zero_mask = arr != 0
44      non_nan_mask = ~np.isnan(arr)
45      # Retrieving index of values that satisfy both conditions
46      non_zero_and_non_nan_values = np.nonzero(np.logical_and(
            non_zero_mask, non_nan_mask))
47
48      osm_ids = pd.DataFrame(np.take(osm_id,
            non_zero_and_non_nan_values[0]), columns=["osm_id"])
49
50      output_filename = filename[:-4] + "_geom_modif_ids.csv"
51      osm_ids.to_csv(output_filename, float_format="%.3f",
            index_label="index", sep=" ")
52
53      print("\n\nThe result has been successfully saved in '" + os.
            getcwd() + "/" + output_filename + "'!")
54
55
56
57  ### Main
58
59  if __name__ == '__main__':
60
61      filename = '
            nls_buildings_multipolygons_nb_geometry_modification.csv'
62
63      get_geom_modif_ids(filename)
```