POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA CIVILE,
AMBIENTALE E TERRITORIALE

Master of science
Geoinformatics Engineering

Towards the evaluation of OpenStreetMap intrinsic temporal accuracy and up-to-dateness using historical data

Author: **Francesco Frassinelli**
Student ID: 872912

Advisor: Professor Maria Antonia Brovelli
Co-advisor: PhD Marco Minghini
Academic Year: 2021-2022

Abstract

Data stored in OpenStreetMap can be more accurate or more recently updated than authoritative sources in some areas, thus requiring to look at other approaches than evaluating OSM comparing it against other databases: methods to perform intrinsic quality analysis are then needed. As this topic is so wide, it has been decided to focus on a specific and less investigated aspect: temporal accuracy. Comparing different areas or features within OSM can provide a relative measurement of temporal accuracy and up-to-dateness, which has been done using standard tools. After mapping existing solutions and evaluating new approaches to gather and process historical data, it has been hypothesized that it could have been worth trying to develop a new software, targeting both OSM contributors and researchers, with a simple interface, reasonable speed and modest technical requirements. This goal has been achieved to a good extent with the development of *Is OSM up-to-date?*, after a long process defined by numerous changes of both approaches and libraries, in an increasingly stronger effort to process larger areas more quickly and more efficiently.

Keywords: OpenStreetMap; intrinsic quality; intrinsic temporal accuracy; Ohsome; GIS; Python

Abstract in lingua italiana

I dati immagazzinati in OpenStreetMap in alcune aree possono essere più accurati e aggiornati rispetto a quelli delle fonti ufficiali; possono richiedere pertanto metodi differenti rispetto al valutare OSM comparandolo ad altri database: sono quindi necessari approcci specifici per effettuare un' analisi intrinseca della qualità dei dati stessi; data la vastità dell'argomento, si è deciso di focalizzarsi su un aspetto particolare e meno considerato di altri: l'accuratezza temporale. Comparare aree e caratteristiche differenti all'interno di OSM può fornire una misura relativa dell'accuratezza temporale e degli aggiornamenti effettuati utilizzando strumenti comuni. Dopo aver recensito le soluzioni già esistenti e aver valutato nuovi approcci per raccogliere e processare i dati storici, si è pensato che fosse giustificato lo sviluppo di un nuovo programma, diretto sia ai contributori di OSM sia ai ricercatori, dotato di una interfaccia semplice e di una buona velocità, con requisiti tecnici modesti: questo obiettivo è stato raggiunto in buona misura grazie allo sviluppo di *Is OSM up-to-date?*, dopo un lungo processo scandito da numerose variazioni sia riguardo l'approccio sia riguardo le librerie, in un impegno crescente teso ad analizzare aree più grandi, in tempi più rapidi e in maniera più efficiente.

Parole chiave: OpenStreetMap; qualità intrinseca; accuratezza temporale intrinseca; Ohsome; GIS; Python

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 OpenStreetMap **3**
1.1 The OpenStreetMap project . . . 3
1.2 Data model . . . 4
1.2.1 Nodes . . . 4
1.2.2 Ways . . . 4
1.2.3 Relations . . . 4
1.2.4 Tags . . . 5
1.2.5 Other entities . . . 5
1.3 Historical data . . . 5
1.3.1 Broken history due to element being recreated . . . 6
1.3.2 Partial history due to historical limitations . . . 6

2 State of the art **7**
2.1 Research . . . 7
2.2 Applications based on OSM history . . . 8
2.2.1 Visualization . . . 9
2.2.2 Statistics . . . 9
2.2.3 Frameworks . . . 10
2.2.4 Data conversion . . . 10
2.3 Database exports . . . 11
2.3.1 Planet.osm . . . 11

2.3.2 ORC files . . . 11
2.4 APIs . . . 12
2.4.1 OSM API . . . 12
2.4.2 Overpass . . . 13
2.4.3 Ohsome . . . 14

3 Research and development history 17
3.1 Exploratory work . . . 18
3.1.1 Tag trends analysis . . . 18
3.1.2 Web application prototype . . . 18
3.2 Historical data and aggregated analysis . . . 19
3.2.1 Multiple criteria for evaluating data . . . 19
3.2.2 Development of the web application . . . 19
3.2.3 Offline aggregated analysis . . . 20
3.2.4 Conferences and papers . . . 21
3.3 Running aggregated analysis in real-time . . . 25
3.3.1 Evaluated options . . . 25
3.3.2 Chosen option . . . 27

4 Final solution 29
4.1 Usage . . . 29
4.1.1 Web interface . . . 29
4.1.2 Command line . . . 33
4.1.3 GIS software . . . 36
4.2 Code . . . 37
4.2.1 GitHub repository . . . 38
4.2.2 Code quality . . . 38
4.2.3 Dependency management . . . 39
4.2.4 Docker . . . 40
4.2.5 Monitoring . . . 41
4.3 Performances . . . 41
4.3.1 Piazza del Duomo, Milano . . . 42
4.3.2 Municipio 1, Milano . . . 43
4.3.3 Milano . . . 44
4.3.4 Città Metropolitana di Milano . . . 45
4.4 Architecture . . . 46
4.4.1 Frontend . . . 47
4.4.2 Backend . . . 47

4.4.3 Caching . . . 48
4.5 Limitations . . . 49
4.5.1 Area of study . . . 49
4.5.2 Relation to GIS software . . . 49

5 Reception from the community **51**
5.1 Respondents . . . 51
5.2 Features . . . 52
5.3 General questions . . . 55
5.4 Possible usages . . . 57
5.4.1 Practical applications . . . 57
5.5 Additional comments . . . 58

6 Conclusions **61**
6.1 Final considerations . . . 61
6.2 Future work . . . 62
6.2.1 Research topics . . . 62
6.2.2 Features . . . 62
6.2.3 Performances . . . 63
6.2.4 Hosting . . . 65

Bibliography **67**

List of Figures **81**

List of Graphs **83**

List of Tables **85**

Acknowledgements **87**

Introduction

This research starts with the following question: OpenStreetMap (which is the most successful crowdsourced spatial database [46]) is a great source of data, but is it possible to learn something about its quality just by looking at its history? Is it possible to provide an integrated and easily accessible solution to both researchers and OpenStreetMap contributors to do that? Could such a solution be a driver towards updating and improving OpenStreetMap data where this is most needed?

These questions are driven by the consideration that OpenStreetMap grew so much lately [110], that in some cases it can be more accurate or updated than official data sources [8], so assuming them as ground truth for quality estimations could be limiting, especially when considering how quickly information can be updated on OpenStreetMap to reflect the current situation.

In order to address these questions, the following goals for this thesis have been established:

1. evaluate the feasibility of estimating the temporal accuracy of an area mapped in the spatial database OpenStreetMap (meaning that the data gets updated in a timely manner), without relying on authoritative data sources, but just by comparing different areas represented in such database combined with their historical evolution;

2. evaluate the technical feasibility of performing temporal analysis with limited computational resources (an ordinary computer with an average internet connection);

3. evaluate whether software solutions tackling similar questions are already available, and whether they allow users familiar with OpenStreetMap to perform basic temporal analysis in a user-friendly manner (meaning that users should not need to write software code or scripts);

4. improve existing solutions, or develop a new one if no existing software satisfies the

previous requirements;

5. suggest new and different technical approaches to further improve how the data is gathered and processed.

1 | OpenStreetMap

1.1. The OpenStreetMap project

OpenStreetMap (OSM) is the most successful crowdsourced spatial database [46], which can be thought as the Wikipedia of spatial data. Users are used to access it using its main web interface by browsing *openstreetmap.org* [109], which can give the false impression that OpenStreetMap is just a map, which can be used similarly to what can be provided by services like Google Maps, while instead that is just one of the possible ways to represent a subset of the OSM database. This difference is crucial, as it is possible to use such data not just to use different styles to render new maps, but as a source for analysis and simulations.

The usage of OpenStreetMap is facilitated by its open access licence, ODbL (OpenDatabase License) [108], which allows anyone to freely use and modify it as long as attribution to the contributors is given. The decision to keep it as open data, together with some intuitive editors and applications, brought a huge number of contributors to the project: over 8 million registered users [113], which uploaded more than 9 billions GPS points, created 7 billion unique nodes, 800 million ways and 8 million of relation between different features, over more than 100 million changesets [110]. That lead to the development of a multitude of different applications, services and tools, that go way beyond the usual base map layer commonly used GIS applications or news websites, allowing a wide number of developers, humanitarian operators, industry and governmental actors to exploit OSM data daily and for a variety of purposes [99].

There are plenty of different features that can be found in such database: from primary features [86], like roads, houses, shop and rivers, to smaller ones, like benches, trees, street lights, and tags that are used to describe the number of steps in a staircase [171], its width, material, accessibility features and such.

1.2. Data model

The most important entities in the OSM data model are: nodes, ways, relations, and tags. There are also entities which model group of changes, comments to discuss such changes or opening new discussions on a specific area which requires attention and other less relevant ones [39].

1.2.1. Nodes

Nodes are simple points, defined by latitude and longitude, expressed using decimal degrees using the WGS 84/EPSG:4326 coordinate system.

Features that can be described as points are individual street laps, bus stops, road signs, entrances of shops, and any feature which has a negligible area occupied, or that occupies an ideal position, like the peak of a mountain. Sometimes even features which occupy a wider area are represented as points, just for convenience or as first approximation, like a shop, but this is usually discouraged [163].

1.2.2. Ways

Ways are an ordered list of nodes (which means that a direction is implicitly specified), that can be used to represent lines composed by multiple segments, or polygons, in case the starting node is also the final one. Way also have a direction, which is useful to define, among others, one-way roads, the position of the side walks (on the left, on the right side or on both sides), and lanes.

Roads, paths, transmission lines, rivers, buildings, and crop fields are valid examples of spatial entities that can be modelled as OSM way.

1.2.3. Relations

Relations are a flexible and powerful entity in OSM, as they are lists of generic elements (nodes, ways, and even relations themselves), where each of them can have a specified role.

A common case for using relations is to express the link between two areas to represent a building with a courtyard, using a polygon with the role “outer” and another polygon inside with the role “inner”. A bus line can also be expressed as a relation, as an ordered group of stops and ways. Another case where relations can be useful are for representing relations between roads, like when it is forbidden for a car to turn on a specific direction

coming from a certain way.

1.2.4. Tags

Tags are key/value pairs, which are used to describe a specific object, like the name of a shop, its opening hours, the maximum speed of a road, or if horses are allowed on a certain path.

Members of the OSM community propose new keys and values frequently, and such proposals are discussed and evaluated with other members, and sometimes they became an officially suggested tag in the OSM wiki-portal, giving life to a crowdsourced taxonomy. *Taginfo* [114] is a widely used service to measure the popularity of a tag, how often it is used, and in which combinations.

1.2.5. Other entities

OpenStreetMap also store additional entities, such as:

changes to describe groups of edits to nodes, ways or relations using a textual description and source of the data (survey, import, aerial imagery, or other);

comments to discuss edits between users;

notes to report issues linked to a specific location, such as missing or outdated data;

tracks to create new paths based on recorded with GPS loggers and similar solutions;

diary to allow the user keep a personal blog on OSM.

1.3. Historical data

Every time a contributor sends a batch of changes (technically referred to as a *changeset* [24]), such changes are applied to the database (if there are no conflicts provoked by some changes made in the meanwhile by someone else) and a changeset object is also created, associated to a timestamp, user ID, and some optional fields, like a description or the source of the data. That allows OSM to keep a history of what happened in the past, which is the main focus of this thesis, but there are some considerations to consider before analysing historical data.

1.3.1. Broken history due to element being recreated

Each node, way, or relation has a unique ID, but that does not mean that they should be considered as permanent identifiers of a specific physical feature is represented by an OSM ID because such entities can be deleted and recreated. That can happen because of a mistake, like a user recreating a shop just because it changed name, or because it has been closed and then opened again. This can be problematic, as there is no link between the old and the new ID, so the history of the new object is incomplete.

1.3.2. Partial history due to historical limitations

OSM underwent a big change in late 2012, when it was decided to migrate from the Creative Commons Attribution-ShareAlike (CC BY-SA) 2.0 license [30] to ODbL [107]; objects related to old contributions that were not released by their authors under the new have been removed from the history file, as the two licenses are incompatible, and the data released under the two licences cannot be mixed for legal reasons. Nowadays, when a user mention the OSM history, is actually referring to the OSM history released under ODbL terms.

This is a minor problem, as the number of elements affected is low compared to the total, since $\sim 1\%$ of the data has not been released under the new ODbL licence [108], and OSM grew enormously in the following years.

2 State of the art

2.1. Research

Being a potentially useful source of geospatial information for many disciplines, OSM has attracted as well an increasing interest from the academic and scientific community [78]. Not surprisingly, the topic which so far has been most investigated by researchers is OSM quality assessment [159], since crowdsourced geographic information suffers by definition from a general lack of quality assurance [55]. OSM quality has been traditionally assessed using the standard quality parameters available for geospatial datasets, e.g., positional accuracy, completeness, logical consistency, thematic accuracy, temporal accuracy, lineage, up-to-dateness, and fitness-for-use [55, 177, 34, 73]. The latter suggests that quality should not be measured in absolute terms, since crowdsourced datasets such as OSM may have different degrees of suitability for specific purposes and users' demands.

These quality parameters have been traditionally measured through extrinsic quality approaches, i.e., by comparing OSM against external reference datasets considered as the ground truth, such as those provided by national mapping agencies and commercial mapping companies. The quality parameters which have been most investigated through extrinsic approaches are positional accuracy, completeness and thematic accuracy, with focus mainly placed (in descending order) on OSM roads [60, 23, 56, 21, 16], buildings [61, 42, 47, 20], land use [41, 77, 45] and points of interest [53, 75]. Overall, the available literature agrees that OSM quality shows very heterogeneous patterns across space, ranging from areas where it favourably compares with authoritative datasets (typically the most urbanized areas) to areas where data is either missing or of poor quality.

However, OSM and authoritative datasets are extremely different by nature, e.g., in terms of their production and update processes which often lead OSM to be clearly more detailed, accurate and complete than authoritative datasets, thus violating the basic hypothesis of using the latter as ground truth [8]. In addition, in many parts of the world authoritative datasets are either missing or not suitable for comparison (e.g.,

because their scale is too course and not comparable to OSM).

OSM history has been investigated by many researchers to achieve several objectives. Quality assessment is by far the most frequently occurring. Ciepłuch et al. [26] developed a set of quality indicators for OSM, which also considered the history and profiling of contributors. Similarly, the intrinsic methods developed by Keßler and de Groot [80] and Muttaqien et al. [100] modelled the quality of OSM objects based on historical information such as the number of contributors and the number of versions. The history of OSM objects and OSM contributors was exploited by Mooney and Corcoran [98] to analyse contributor patterns in seven cities around the world. Gröchenig et al. [57] developed an intrinsic approach for the assessment of OSM completeness through the analysis of community contributions over time. A novel approach for improving the positional accuracy and completeness of the OSM road network using the OSM history was proposed by Nasiri et al. [101]. Barron et al. [17] developed a comprehensive framework for OSM intrinsic quality assessment, which includes more than 25 methods and indicators exclusively based on OSM history.

Availability of OSM history was also exploited for the intrinsic analysis of the temporal evolution of specific OSM objects. For example, the growth of the OSM road network was analysed for countries such as Ireland [29] and Germany [102], and cities such as Beijing [185] and Ankara [59]. Barrington-Leigh and Millard-Ball assessed the completeness of the OSM road network at the global level (finding a value of about 83%) by also studying its historical growth [16]. A work by Tian et al. [172] analysed the evolution from 2012 to 2017 of OSM buildings in China. Jokar Arsanjani et al. [77] modelled OSM evolution in Heidelberg through a spatio-temporal analysis of OSM contributions, while Minghini et al. [91] performed an intrinsic analysis of OSM nodes evolution in Dar es Salaam, highlighting clear spatio-temporal patterns driven by a community mapping project. Using intrinsic quality indicators based on OSM history, Sehra et al. [159] assessed OSM evolution in India. OSM historical information was also used as a means to characterize the contributors' response in the aftermath of natural disasters, e.g., in terms of frequency of updates and types of contributors (novice vs. experienced OSM users) [35, 94, 15].

2.2. Applications based on OSM history

There is a wide variety of software which use OpenStreetMap history, which uses different approaches and have different scopes. A comprehensive review has been made in 2019 by F. Frassinelli and M. Minghini [92], which has been used as starting point for this section.

2.2.1. Visualization

In terms of visualization, the easiest way to access OSM history is through the OSM website [109], which, in addition to map browsing, routing and other features, also offers historical information for each OSM object selected. Achavi [3, 4] is a JavaScript web app leveraging the Overpass API to display OSM changes happened in any user-specified time frame. Based on the OSM Full History Planet File, OSMatrix [158] offers web-based visualization of OSM spatio-temporal quality indicators using a hexagonal grid. This web application is no longer maintained and was recently replaced by OSM History Explorer [106]. Based on the Ohsome platform [105], it offers grid-based visualizations of the density of a number of predefined variables connected to OSM objects (number of buildings, length of different types of roads, etc.) for any given region and at any specific time (month) in history. OSM Changeset Analyzer (OSMCha) [124] is a web application to validate suspicious OSM changesets based on several criteria such as location, comment, date, number of modified objects, user, source, and editor. Another popular web application to analyze changesets is Who did it? [184]. OSM Deep History [115] and OSM History Viewer by PeWu [116] are web applications providing simplified access to the history of single OSM objects. A similar web application is Visualize Change [178], which graphically shows the evolution of roads and buildings over time. OSM History Viewer (osmrmhv) [129] visualizes the changes made in single changesets, using different colours for different actions (delete, modify, create) and analyses the history of OSM relations, while Show me the way [128] is a popular near-real-time graphical representation of the latest OSM edits. Using the OSM Full History Planet File, OSMvis [95, 96] offers a number of visualizations to explore the generation, modification, and use of OSM through the methods of information visualization. Finally, OSM Latest Changes [117] lists the latest changesets happened in any given region and highlights the ones corresponding to the object selected on the map.

2.2.2. Statistics

Statistics on the OSM history can be found foremost in the OSM wiki [166], where multiple plots are displayed showing OSM evolution in terms of both objects and users. Another rich source of information on OSM evolution is OSMstats [130], which offers statistics and graphs about OSM users, objects and changesets both at the global and country-level scales. Statistics on the time, frequency, place, and type of mapping for each OSM user are provided by the web application How did you contribute to OpenStreetMap?

[64]. For any user-selected area, the QXOSM web application [150, 6] provides statistics and plots of different indexes for both objects and users, computed from the OSM history. OSM Tag History [118] produces graphics on the evolution in the usage of specific tags over time and the comparison between selected tags on a global scale. Brave Mappers [19] creates colourful graphics and statistics showing OSM contributors' activity in a specific area. iOSMAnalyzer [17] is a tool for intrinsic OSM data quality analysis, which takes the Full History Planet File as input and generates statistics, maps and diagrams to assess the quality of selected areas. A number of web applications combine visualization and statistics based on OSM history. Examples are OSM Analytics [120], which describes the evolution of OSM objects in a given region and time frame and offers as well a side by side comparison of the OSM map at different points in time, and OSM Live Changes [111], which provides near-real-time visualization and statistics of OSM edits in the whole world. The Oshome Dashboard [104], based again on the Oshome platform [105], allows analysing the OSM history based on advanced filtering and grouping functionalities on keys, values, region of interest and time, generating plots and returning results in a JSON or CSV file.

2.2.3. Frameworks

Two frameworks for the analysis of OSM history based on the Full History Planet File are worth mentioning. The first is the already mentioned Ohsome platform [105], a powerful big data framework leveraging the OpenStreetMap History Database (OSHDB) [151] to offer researchers fast data access and flexible analysis methods. Similarly, OSMesa [125] provides a rich collection of tools to simplify the analysis and processing of OSM history.

2.2.4. Data conversion

There are finally many applications for conversion of OSM history data. The EPIC-OSM framework [142] processes the OSM Full History Planet File with predefined queries, called *questions*, and extracts descriptive analytics to understand community contributions and collaboration in OSM. The OSM PBF Foreign Data Wrapper [137] allows querying OSM history stored into a PBF file directly from PostgreSQL. Indexing is not supported and queries can be slow, but this issue can be avoided by importing the history into a native PostgreSQL database. OSM Parquetizer [18] allows instead to convert the OSM Full History Planet File into a Parquet file, suitable for big data analysis within the Hadoop ecosystem. Osmium [127] is a powerful tool to extract and convert the OSM Planet file, with partial support for history files. It is based on a C++ library (libosmium) which can be used by developers to create new tools; Python bindings are also provided.

2.3. Database exports

Database exports (also called *dumps*) are usually meant to be loaded into a database or to extract a smaller area for further processing, as using them as they can be computationally expensive, since no indexes are provided. Having a spatial index, as well as indexes on tags, is crucial to be able to select a portion of space efficiently, as it is needed when trying to assess the intrinsic quality of a given area.

2.3.1. Planet.osm

The most common database export is called *Planet.osm* [138], which is the only non-experimental export provided by OpenStreetmap, and includes a snapshot of the most recent data. That includes limited information about the history of each feature: the version number (an integer that is increased each time the feature has been modified) and the timestamp of the version [119]. As a result, it is not possible to know how many revisions have altered the geometry without changing its tags (for example, a node being moved to a more accurate position), which tags or geometry a feature had previously, or get deleted features.

To have a better overview about the temporal evolution of the database, an experimental, but widely used, type of export is made available: the *OSM full history file* [139].

It is delivered as a compressed XML file (110 GB at the moment of writing) or as a PBF file[1] ($\sim$ 60 GB), but once imported in Postgres its size explodes to more than 1 TB and can require days to be imported [123].

Some efforts have been made to simplify the synchronization, like the project docker-osm by kartoza [38], which rely on the imposm [68] tool. Another common tool frequently used is osm2pgsql [122].

It is possible to use the Planet.osm PBF file and rely on it entirely, while providing a database interface. There has been developed a Postgres FDW for OSM PBF files [137], but is only meant to facilitate import into Postgres, as no index is created.

2.3.2. ORC files

Some companies maintain (or used to maintain) an alternative version of the Planet file, stored in a ORC columnar file format. One of them was Skobbler/Telenav [121], while the

[1] PBF stands for Protocolbuffer Binary Format, which is a file using the Protocol Buffer format [157], developed by Google and meant to provide a more efficient alternative to serialize data

other is Amazon, which maintains an updated set of ORC files (planet, history, changesets) to be downloaded as a whole or queried using its AWS services [112]. Rely on AWS services can be challenging, as they require using a credit card, are strictly dependent on the AWS ecosystem (S3 has to be used), there is only one available datacenter to use AWS Athena (located in Virgina, USA), handling storage and egress costs require a non-negligible amount of knowledge, and a developer investing its time into developing a service based on such platform need to rely entirely on a closed source hosted solution, with limited understanding of how the underlying service works and the impossibility of adapting the software based on new needs or requirements that can only be satisfied by modifying the source code.

2.4. APIs

Rely on API instead of database can be often limiting, as the developer abilities are limited by the query language and by the finite number of options and parameters, thus reducing the amount of possible queries that can be made.

2.4.1. OSM API

OpenStreetMap has an official API [10] that can be used both to read and to write data. It also provides an endpoint for getting the history of a single element [12]. This API comes with various limitations regarding historical analysis that have been discovered.

Maximum number of nodes

OSM API returns an error when the area requested contains more than 50000 nodes. Such a problem is not easily solvable by making multiple requests, since there is no way in advance to know how many nodes an area has. An algorithm which spit the requested bounding box recursively based on the server reply could be used, but with an important overhead and potentially high number of requests.

Rate-limiting/ban mechanism

During the testing of the endpoints, it has been found that the server probably has some rate-limiting capabilities to prevent usage abuses, but it has not been possible to discover precisely which policy has been established. It has been experienced a temporary ban on the IP which generated too many requests in a brief amount of time, in the order of hundreds of requests over a few minutes.

No bounding box history

It is not possible to get the history of all the features included in a given bounding box within a single request, as the history is provided for a specific feature only. Such limitation would then imply a lot of overhead for potential API users interested in history-related analysis on multiple objects.

It has been found that performance could still be reasonable for areas with $100 \sim 1000$ features using HTTP pipelining[2], by relying on HTTP 1.1 only (HTTP pipelining has been introduced with HTTP 1.1, and the OSM API server does not seem to support HTTP 2 or later versions). The maximum number of requests that can be pipelined is 100. This capability, as well as the details described before, is not documented on the OSM wiki or by the OSM API policy: such information have been discovered empirically, by querying the endpoint with different methods and parameters.

Pipelined requests greatly reduce the risk of a temporary ban due to an excessive number of requests, hinting that the rate-limiting mechanism monitors the number of connections over time, but not the number of requests.

Policy violations

In addition to such technical limitation, the OpenStreetMap Foundation, which maintains the service, explicitly discourage using the API for read-only purposes [11]. One can argue that tools that provide insights on the status of a selected area are also probably used by users that are interested in spotting bad or outdated element to fix them, but as there is not a well-defined border from a merely technical perspective, it is reasonable to not flood OSM servers with high volume of read-only requests.

Self-hosting and alternative providers

There are no known alternative servers hosting the OSM API service, nor there is any documentation on self-hosting such service, which makes the possibility of running the service on premises not feasible.

2.4.2. Overpass

OverPass API [131] is probably the most widely used open-source API for read-only queries targeting OpenStreetMap, which address the biggest problem of using OSM API

[2]HTTP pipelining consists in sending multiple requests over a single TCP connection, without waiting for the corresponding responses

for analysis (they were not meant for that), and it has its own Overpass Query Language [132]. Such language can express various spatial operations, attribute filtering, exploring relations recursively and even implements loops in certain cases. Filtering based on attributes works on the national scale if the number of results is on the order of the hundreds of features. It also has some features to fetch the history of an element since v0.7.55 [164] (released in 2018). Overpass API can be self-hosted, and it also served by various OSM community servers [131]. Even if promising, this API has some drawbacks too, analysed below.

No bounding box history

Similarly to OSM official API, OverPass API cannot get the history of all the features within a bounding box.

Getting the history of more than one element implies asking a list of features matching a spatial criterion first, and ask the history of each of them individually. This is impractical from a performance perspective (even asking for $10 \sim 100$ of features makes the processing so slow that a timeout is reached before the result is computed), making it even less suitable than the original OSM API for such purpose.

2.4.3. Ohsome

The Ohsome API [181] is a more recent effort to provide an open-source API to address history-related analysis, started at the end of 2017 [69] which added an endpoint to get the history of multiple elements a year later [67], ended up in the following release, Ohsome API 1.0.0 of June 2020 [156]. Ohsome API is the only one which allow fetching the history of a set of features within a bounding box. Performance is good (few seconds for $1000 \sim 10000$ features) if details about how tags and geometry changed over the history are not requested.

The Ohsome API can be set up on a server using the OSHDB [152], which internally uses a custom database containing all OSM data, or it can be used by relying on the only official public instance [170]. The official public instance does not require any authentication. The group developing Ohsome can be easily contacted by mail or on GitHub; replies happen timely and the developers always shown a positive and welcoming attitude when contacted regarding issues, features, or performances inquiries.

Uptime

Its availability is monitored by *status.ohsome.org* [167], which reports values around 97% uptime at the time of writing (≈ 45 minutes per day in average). This is a decent result for a free service hosted by a university, but is below what services like Overpass (thanks to being hosted by multiple entities) or OSM API provide. Some downtimes last for hours or days, and none of them is planned and communicated to external users.

No user data

Ohsome do not provide information about the number of unique users which have modified a feature [5], nor it is possible to get any user ID or username due to privacy policies [126].

3 | Research and development history

The possibility of estimating the quality of OSM data using its history begun in 2017. Two different approaches have been evaluated: analysing the evolution of OSM tags for a given class of features (such as common life-cycles patterns for shops), and analysing OSM features based on when they have been edited during their existence (such as last edit, creation time, or number of revisions). The second approach appeared more promising, and produced two methods to run the analysis: a web app, capable of showing and classifying individual map features over a small area ($\sim$ 1km^2) in few seconds, and a software to classify wide areas ($\sim$ 100 000km^2) requiring various hours of computation. Given the wider target the web app solution could address, thus allowing to have a greater impact on OSM mapping activities and *gardening of the data*[1], an improved version of the web application has been developed, in the tentative of providing a solution capable of doing aggregated analysis over wide areas, but in a matter of seconds, without requiring the user to set up complex software.

A criterion which drove the development process, has been the cost for producing such analysis service at a reasonable price (less than 10 €/month, which is the price of a small virtual private server), to make it economically sustainable (as the service is provided for free), allowing to run it also locally on a regular PC. That, together with time constraints for the computation, affected the whole development, pushing toward choosing efficient solutions, ranging from the usage of in-memory SQLite databases to handle data, to fine-tuning network communications using HTTP pipelining and compression, from reducing the memory usage by avoiding storing intermediate results via streaming techniques, to experiment with streaming JSON parsers and using different caching techniques using the Redis database.

[1]The expression "data gardening" refers to taking care of the data regularly

3.1. Exploratory work

3.1.1. Tag trends analysis

An experiment to figure out if some trends could be established by looking at the evolution of OSM features over time has been developed after the first prototype of *Is OSM up-to-date?*. In order to find potential trends in the data, the full OSM history has been downloaded, all the data outside Italy have been discarded (in order to achieve a manageable sized file), and then processed using the Python bindings of the Osmium library, together with Spatialite.

Strong trends have been observed, where a tag has been frequently added after another tag (but not the opposite). For example, the key *tourism* has been added after *opening_hours* 77% of the times, and the sequence *opening_hours*, *parking*, *tourism* appear in this precise order 99% of the times.

Many trends are influenced by the effort of the community in adding some specific tags (like *wheelchair* or *wikidata*), or they are of related to public transport.

Even if the results were interesting, such kind of analysis has been abandoned, as it was found more suitable for doing *semantic accuracy analysis* than for temporal quality assessments.

Code has been published as a GitHub Gist[2] as *osh2sqlite.py* [1].

3.1.2. Web application prototype

A simple web prototype, named *Is OSM up-to-date?*, has been developed and published on GitHub during August 2017. Such web app provided an alternative map to explore OSM data, classifying the features of a given area by their latest update.

This initial version was relying on Overpass API, via a tool shipped together with Spatialite[3], named *spatialite_osm_overpass*. Such tool fetches data from Overpass for a given bounding box, and transform the result into a SQLite database. The prototype relied on a relatively new SQLite feature, the JSON1 extension introduced less than two years before in SQLite 3.9 [165], to build a JSON object within an in-memory database, that was then sent to the web front-end, parsed and represented over a Leaflet map.

Only nodes were represented. They were coloured based on their up-to-dateness: warmer

[2]GitHub Gist is a GitHub service to store and share snippets of code

[3]Spatialite is the SQLite extension for handling spatial data

colours indicates a recent edit, while cold colours an old one. This simple criteria allowed the user a measure of how far in the past a node has been updated, relatively to its near nodes included in the selected bounding box. The bounding box used for such analysis was set to the current map viewport, while geometry changes were ignored, as the tool only evaluated tag changes.

The code of the prototype can be seen in the first commit of the repository *frafra/is-osm-uptodate* on GitHub [70].

3.2. Historical data and aggregated analysis

The initial good result, obtained with the web prototype for evaluating if data have been recently updated, pushed the research into two different directions: establishing multiple temporal criteria and aggregate data by spatial proximity to evaluate larger areas instead of individual features.

3.2.1. Multiple criteria for evaluating data

The following criteria have been considered:

first edit showing when the feature has been created;

last edit showing when the feature has been modified last time;

revisions as the number of edits which changed the attributes of the feature (thus excluding changes affecting the geometry of the feature only);

update frequency as the number of revisions made over a time span ranging from the creation of the feature until today;

contributors as the number of unique users that edited the attributes of the feature.

3.2.2. Development of the web application

After having developed a working prototype, the web application got improved, by adding the previously describe criteria for analysis on individual feature, which required fetching and handling a more diverse set of data, compared to a single request to the OSM or Overpass API for getting some insights about the features included in a specific bounding box.

Data fetching and parsing

Due to limitations of the Overpass API, OSM API had to be used for retrieving historical data, thus the tool *spatialite_osm_raw* replaced *spatialite_osm_overpass* used in the initial prototype. As an additional request for each feature is needed, to get information about its history. As the process is too slow, HTTP 1.1 pipelining technique has been used to speed up the data fetching.

Docker and CI testing

A major improvement has been the adoption of Docker as container technology, to easily bundle the software and deploy it to the servers or locally. These improvements simplified the development of continuous integration pipelines, which have been used to run basic tests on each new commit of the software, to spot potential regression and fix them before they end in a successive release without noticing.

Other improvements

Support for ways has been added too, alongside the existing support for nodes. Some improvements on the interface have been implemented after the software has been shown to the OSM community, such as a viridis palette[4] (which replaced the warm/cold colour classification for the features) and a greyscale OSM base map, to avoid confusion between the colours used to classify the features and the ones used by OSM standard tiles.

3.2.3. Offline aggregated analysis

Aggregated analysis has been implemented outside the web application, given the vast amount of time and resources needed. The criteria have been further extended, to compare areas such the ones generated by a hexagonal grid or the ones defined by administrative local boundaries, by considering the average of the given criteria for all the features within each area. In addition to that, some other criteria have been considered, such as the total number of contributors by square km, representing the density of unique contributors over an area.

Data fetching and parsing

Data is downloaded from the OSM planet server, where the full history is available as a PBF file. The data is then cropped over the study area, to reduce the amount of time

[4]The viridis palette is a colour-blind friendly palette meant to present continuous variables

and space required for the successive steps. The resulting PBF file is the converted to a custom defined SQLite database using *osh2sqlite.py* [1], which relies on the Osmium library, which has an official binding for the Python interpreter. Such database contains only a subset of the initial information (the one which is relevant for the analysis), which is easily processed by any software compatible with SQLite.

Analysis

After the database is generated, it can be loaded directly into a GIS software, as QGIS, or processed using a SQL script, to produce the aggregated statistics, before being loaded into a GIS. The latter choice has been preferred [2]as it requires less user interaction and relies entirely on SQLite and Spatialite capabilities of processing data in transaction, which makes it quick and robust.

3.2.4. Conferences and papers

The results of this work and the analysis has been shown during various national and international conferences, and some papers have been published as well.

FOSS4G-IT/Merge-IT 2018 – Turin

The Italian FOSS4G conference happened within the bigger Merge-IT conference in Turin, Italy, the 28th of May 2018, as a dedicated geospatial track. A presentation with the title "*Un approccio open source per la valutazione intrinseca di accuratezza tematica, accuratezza temporale, aggiornamento e lignaggio di OSM*" has been presented by Francesco Frassinelli, co-authored by Marco Minghini and prof. Maria Antonia Brovelli [51]. Such presentation shown the current status of the web application as well as aggregated statistics on every Italian city and town. Statistics about tag trend have been shown too.

The aggregated analysis hinted to some strong correlations between the density of unique contributor and highly populated areas (figure 3.1), as expected, and that most of the data in Italy stored in OSM had a low number usually: just a fifth of the towns have an average below 2 revisions.

FOSS4G 2018 – Dar es Salaam

In June 2018, a quality assessment report of the Dar es Salaam area (Tanzania), which has been the venue for the FOSS4G 2018 conference, has been published, with the title:

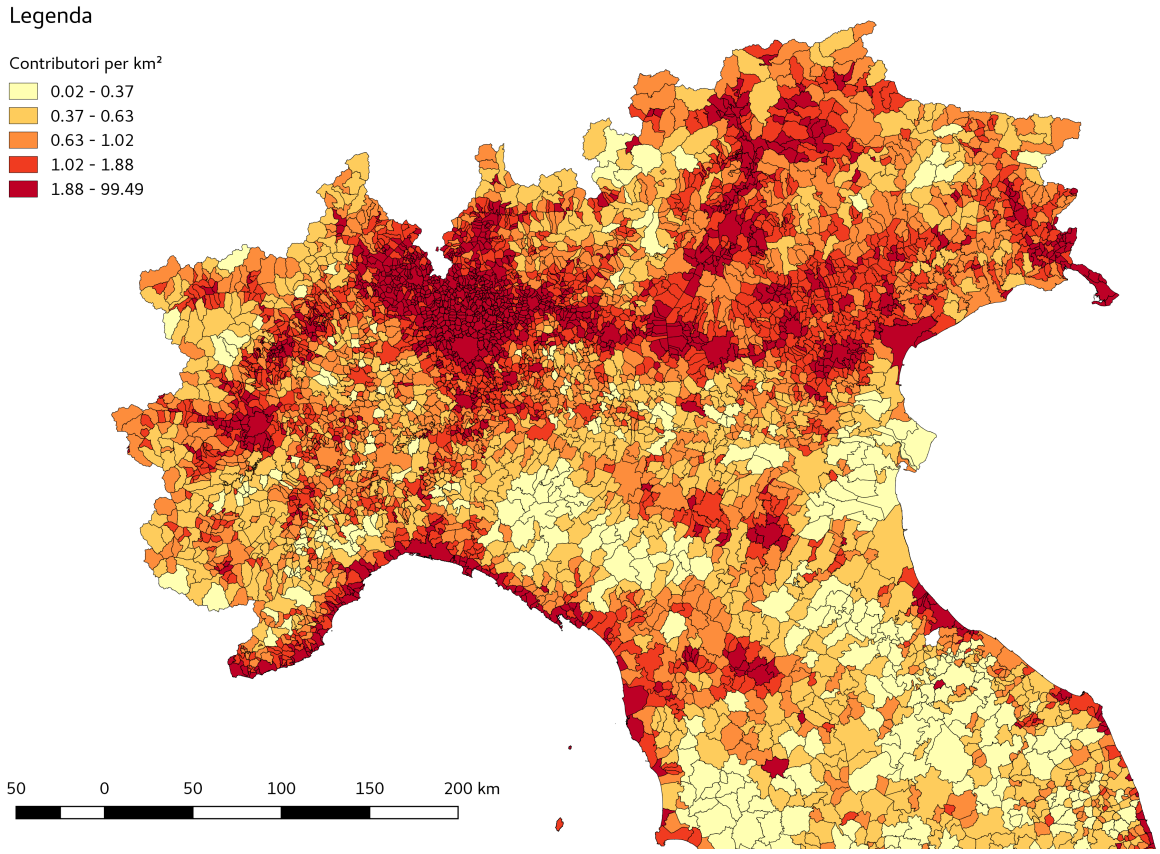


Figure 3.1: FOSS4G-IT/Merge-IT 2018 – North Italy, contributor density

"An open source approach for the intrinsic assessment of the temporal accuracy, up-to-dateness and lineage of OpenStreetMap", by Marco Minghini, prof. Maria Antonia Brovelli and Francesco Frassinelli [91].

Such report shown the evolution of the mapping processes in the area, showing that the vast majority of nodes in the city centre of Dar es Salaam were created in 2015, when the Dar Ramani Huria project was initiated. The mapping activity then gradually continued towards the city outskirts in 2016, 2017 and 2018, the latter clearly showing an increased mapping attention on the northernmost area as well as on more or less isolated areas on the west/south-west side of Dar es Salaam. In particular, the set of cells departing along an almost horizontal line from the city centre to the west side of the city correspond to the location of the towers carrying electricity cables (tag *power=tower*). The average date of last edit of nodes confirms the same trend. In fact, only a small portion of the nodes created before the end of 2015 was later updated, while mapping in 2018 focused almost exclusively on the peripheral areas.

7 different maps with aggregated data have been generated and described, using different

criteria, such as node and contributor count, average of creation time, last edit, number of contributors, update frequency and versions.

The report has been cited 6 times, according to Google Scholar.

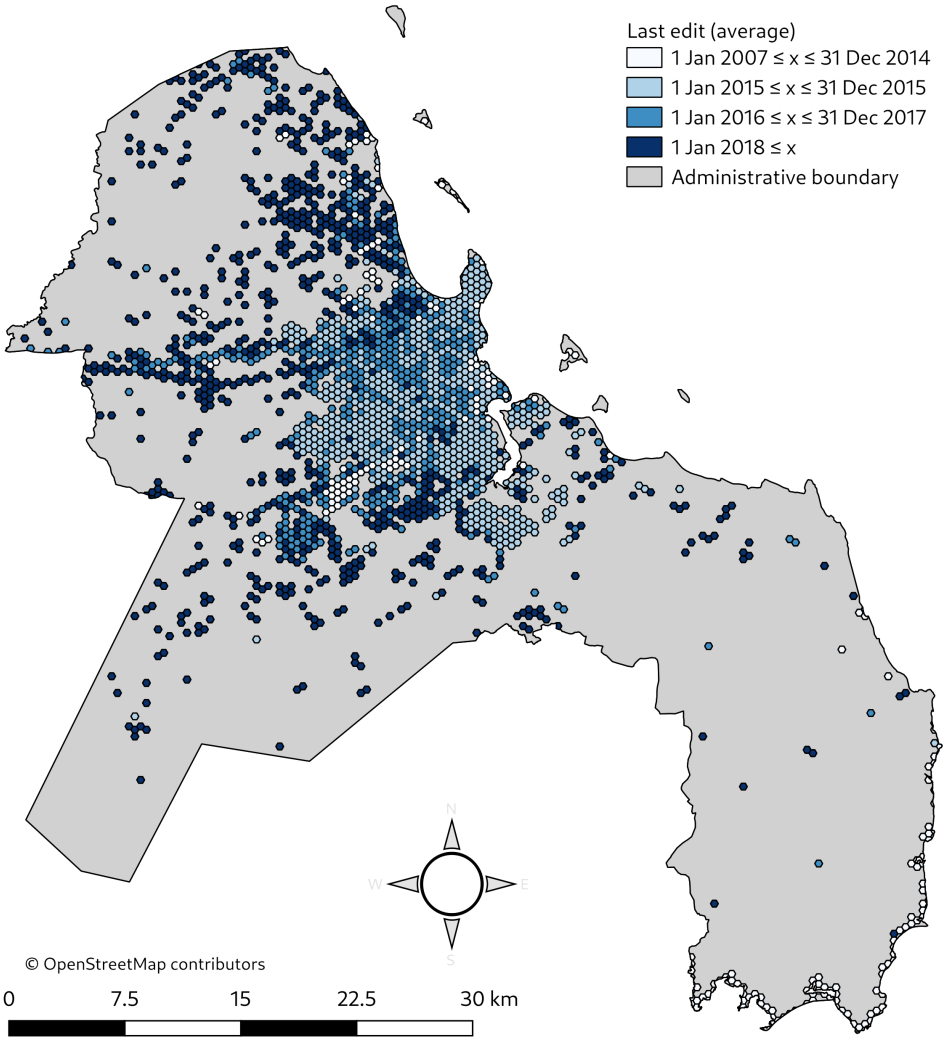


Figure 3.2: FOSS4G 2018 – Dar es Salaam, average last edit

The 31st August 2018, the result of the report has been presented at the FOSS4G conference in Dar es Salaam.

SOTM 2018 and SOTM 2019

The software has been adapted to analyse the city of Milan and the results have been shown at the academic track of the State of the Map conference (the annual international conference of the OpenStreetMap community) in Politecnico di Milano, the 29th of July. The presentation title was *"Intrinsic assessment of the temporal accuracy, up-to-dateness,*

lineage and thematic accuracy of OpenStreetMap", presented by Francesco Frassinelli, co-authored by Marco Minghini and prof. Maria Antonia Brovelli [49].

In such specific context, the personal knowledge of the examined area allowed to suppose various interpretations to the shown results. A strong correlation has been found between the number of revisions and busy roads, as well as high number of contributors in areas where mapping parties happened. It has also been observed that update frequency is mostly homogeneous across the city, with the suburbs being slightly more updated; that could hint that the city centre of Milan is mostly complete from a mapping perspective and new features are created in the suburbs, where there could be more features that still need to be added to the OSM database.

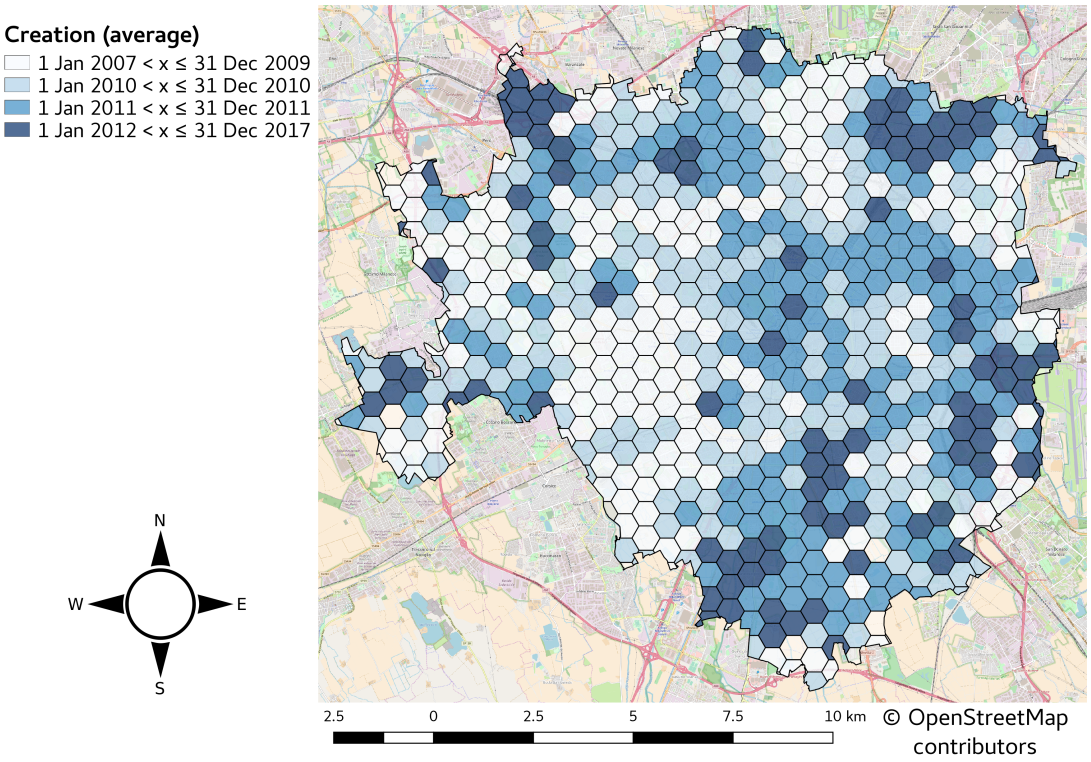


Figure 3.3: SOTM 2018 – Milan, average creation time

The data have then been further used to find spatial correlation patterns using QGIS Hotspot Analysis plugin; the results have been presented at the academic track of SOTM 2019 in Heidelberg, Germany, with a report named *"Intrinsic assessment of OpenStreetMap contribution patterns through Exploratory Spatial Data Analysis"*, by Marco Minghini, Daniele Oxoli, Francesco Frassinelli and prof. Maria Antonia Brovelli [93].

Peer-reviewed paper (September 2019)

A peer-reviewed software paper titled *"OpenStreetMap history for intrinsic quality assessment: Is OSM up-to-date?"*, by Marco Minghini and Francesco Frassinelli, has been published in "Open Geospatial Data, Software and Standards" [92].

The paper reviewed the existing software for handling, showing and analysing OpenStreetMap history, comparing them to the *Is OSM up-to-date?* application, and it has been cited 35 times, according to Google Scholar.

3.3. Running aggregated analysis in real-time

The final goal for this research was to make the web application converge with the methods used for aggregated analysis. This has proved to be a major technical challenge, as the needs of a web application (which is assumed to show results with little delay) and the ones of spatial analysis over larger areas (thus requiring way more data to fetch and handle) are in conflict. Exploring different solutions and approaches, running benchmarks with different parameters and experimenting with different techniques allowed to move towards such result.

An important simplification can be splitting the globe into a grid, which is needed in case of pre-computed statistics, but can also make caching easier. A hexagonal grid would better represent reality, but is more complex to handle compared to a square grid, since APIs commonly used require a rectangular bounding box, and spatial operation usually simpler when using a square grid.

3.3.1. Evaluated options

Pre-computed statistics

The idea behind pre-computed statistics is to have a long-running task which is executed regularly (like once every week or month), which stores just the result of the analysis for each cell or tile of the globe. Such approach is based on the following assumptions:

- the criteria and methods used satisfy the most common and interesting user needs, as the original non-aggregated data is discarded and additional statistics cannot be computed; for example, if no standard deviation has been computed in advance, there is no way to compute it afterwards, except by implementing a software change and run the computation again;
- aggregation of aggregated data can be performed: it should be possible to group

statistics together, to produce data to be displayed at lower scales; for example, aggregating two bounding boxes represented by the averaged revision number can be done if the information about the number of features within the two bounding boxes is stored; otherwise the total average cannot be computed.

By relying on square web tiles [173], it can be easily computed that the number of tiles that should be computed at a zoom level 18 (scale $\approx 1:20000$[5]) is large $4^{18} = 68719476736$ [186]. If we assume that the Earth surface 71% covered by water and nodes with attributes on water are very rare, the number of tiles would drop to $4^{18} \times (1-71/100) \approx 20000000000$. The storage space needed to store 5 16-bit precision float numbers (4 bytes each) for all the terrestrial tiles would then be $20000000000 \times 5 \times 4$ bytes ≈ 400 GB at least. This does not seem very efficient, since the whole OSM history is ≈ 100 GB: four times smaller than such a hypothetical file containing pre-computed statistics. Such difference could be explained by the fact that wide areas of land in OSM have no data; it could be that the final result would be much smaller, if the data store in the resulting file are tightly packed. Handling a regular update, conversion and indexing using a custom format could still require an important amount of computational resources when working on such a big dataset.

Database conversion

Storing the data in a Postgres/PostGIS database could be an alternative, but storage requirements are over 1 TB and the conversion require some days using osm2pgsql [123]. There are alternative methods to import data, such as using a Postgres Foreign Data Wrapper (FDW) for reading OSM PBF [137], but storage requirements are assumed to be similar. Conversion of PBF files to a database is also made by OSHDB [152], which powers Ohsome. Avoiding conversion of planet files (PBF or BZIP2-compressed XML) in the first place would greatly help, but as they do not have a spatial index, this should be generated first. As there is no ready available solution for that and the development of such system would require an additional effort, such solution has then been discarded.

Cloud services

AWS Athena [7] is a service provided by Amazon. It has been evaluated to see if it could have been a viable solution. Amazon also has a copy of the OSM planet history file, which can be used for queries. Running joins is not efficient (probably due to not having

[5]The scale of an individual varies when using different screens, resolutions or when the tile renders data at high latitudes, since the *Web Mercator projection*, commonly identified as EPSG:3857, is a variant of the Mercator projection

indexes), and running complex queries require splitting the database into multiple pieces, which seems impractical for the given purpose, so it has been discarded.

Fetching data from external public API

Benchmarks between the different methods to retrieve historical data on a larger scale from publicly available APIs have been performed. The area considered is centred across "Città Studi" in Milan and is 1 KM^2 wide. Over such area, Overpass with batched queries needs 3 minutes, OSM API + HTTP 1.1 pipelining 30 seconds, while Ohsome API just 5 seconds. Ohsome would be much slower in case the tags of the previous versions are requested or ways and relations are fetched too, but as the focus of the current research does not focus on these aspects, Ohsome seemed the most promising solution. Ohsome API can also return a GeoJSON object, which could be fetched directly from the browser, but the computation would require more time, as the load would be put on the client instead of an intermediate server. The result could not be cached and shared across multiple clients.

Techniques such as caching, parsing and processing while fetching, and data compression can be hugely beneficial, as the data should be retrieved over the network.

3.3.2. Chosen option

Ohsome has been then chosen as data source for aggregated analysis from version 1.6, combined with a caching mechanism from version 2.0, to further improve performances and reduce the load on the Ohsome API server.

Displaying data efficiently

The software initially displayed all the nodes as different features on a map. This approach does not work when hundreds of nodes are displayed on the screen, for two reasons: nodes overlapping and client performances (both bandwidth and rendering).

The problem has been initially addressed in version 1.8, by using a popular extension for Leaflet, called *Leaflet.markercluster* [82], which shows a single circle for each group of close nodes. The colour of such circle varies from green, to yellow and orange, based on the number of nodes in the group. This is useful for a heatmap-like visualization, but it clashes against how colours are used in the web app, as colours represent a quality indicator. Altering how the colour is assigned to group, it is possible to use the same criteria.

Using Leaflet.markercluster is acceptable until thousands of nodes are shown on the screen. It has been tested the possibility of generating PNG images, each representing a group of nodes, and serve them as tiles, following the common Tiled Web Map convention [173]. Such a test ended successfully, so the approach has been adopted to aggregate data on a larger scale.

Serving data efficiently

The software moved away from using Hug [66], due to the inability to stream data, and adopted uwsgi [175] with Flask [43]. That allowed serving static files independently of the Python version and run multiple Python processes and threads.

During the development of version 2.0, which added support for tiles, this approach shown some limitations, as each tile is generated from a different HTTP request, so dozens of concurrent requests are produced by a single client. Since it is not possible to have many threads without incurring in performance penalties due to the Python global interpreter lock and memory usage scales linearly with the number of processes spawned, the software stack for the server had to change again.

Version 2.0 and above use aiohttp [14] combined with gunicorn [58], which allowed to fetch and serve many tiles concurrently with a lower latency and resources.

4 | Final solution

The main result of this research has been the development of the *Is OSM up-to-date?* [71, 50], as well as its deployment as a publicly available service over the web for free, maintained, updated and developed for almost 5 years.

4.1. Usage

The software consists can be used with its web interface or by connecting a GIS software to it. In the latter scenario, an external software can further analyse and process the GeoJSON produced by fetching the historical data over a defined bounding box using the available API, or by relying on the ability of the software to produce raster tiles.

4.1.1. Web interface

The web interface is made by a map and a bar. Data is automatically loaded when the user lands on the page for the first time, and it is updated when the boundaries or the settings changes.

Map

The map shows the current area of interest, and it is made by:

- the standard OpenStreetMap **base map**;
- the **nodes** (if any), rendered individually as coloured circles at zoom level 19 (figure 4.1), or as group between level 17 and 18 (figure 4.2), or as tile between level 12 and 16 (figure 4.3);
- the **legend**, placed in the top-right corner, constituted by:
  - a **colour bar** representing the colour range, with the minimum and maximum values placed at its extremes;
  - a **slider** to convert the base map to greyscale, which is meant to increase

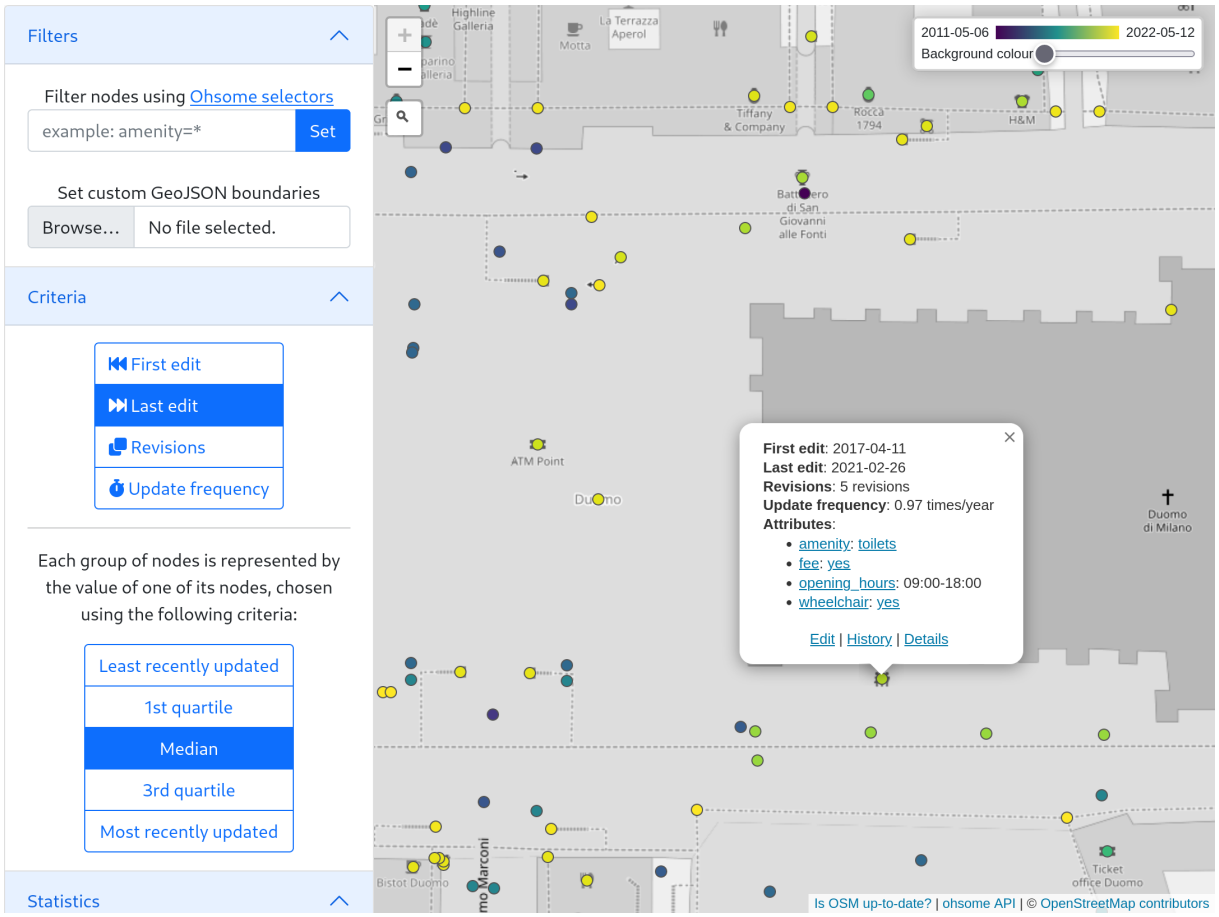


Figure 4.1: Web app – Zoom level 19, individual nodes with pop-up

the contrast between the nodes and the background; the position of the slider defines the proportion of the conversion, which goes from full greyscale (default setting) to full colour;

- **zoom controls**, placed in the top-left corner;
- **search button**, placed below the zoom controls, which opens a search bar with autocompletion when pressed/clicked;
- an animated **loading spinner**, which appears when the map is updating;
- **attribution**, mentioning OpenStreetMap contributors, the Ohsome project and *Is OSM up-to-date?*, linked to its GitHub repository.

Clicking or tapping on a node opens a pop-up, shows the computed values of the node (first edit, last edit, revisions, update frequency) as well as the node attributes (fetched asynchronously from OpenStreetMap API). Links to the OpenStreetMap wiki portal are automatically generated for all the keys and values; a regex is used to avoid creating links

to uncommon values that are not expected to have their page in the wiki.

The pop-up also has links to the web OpenStreetMap website, to make changes using the editor, get the full history of the node, or get more details.

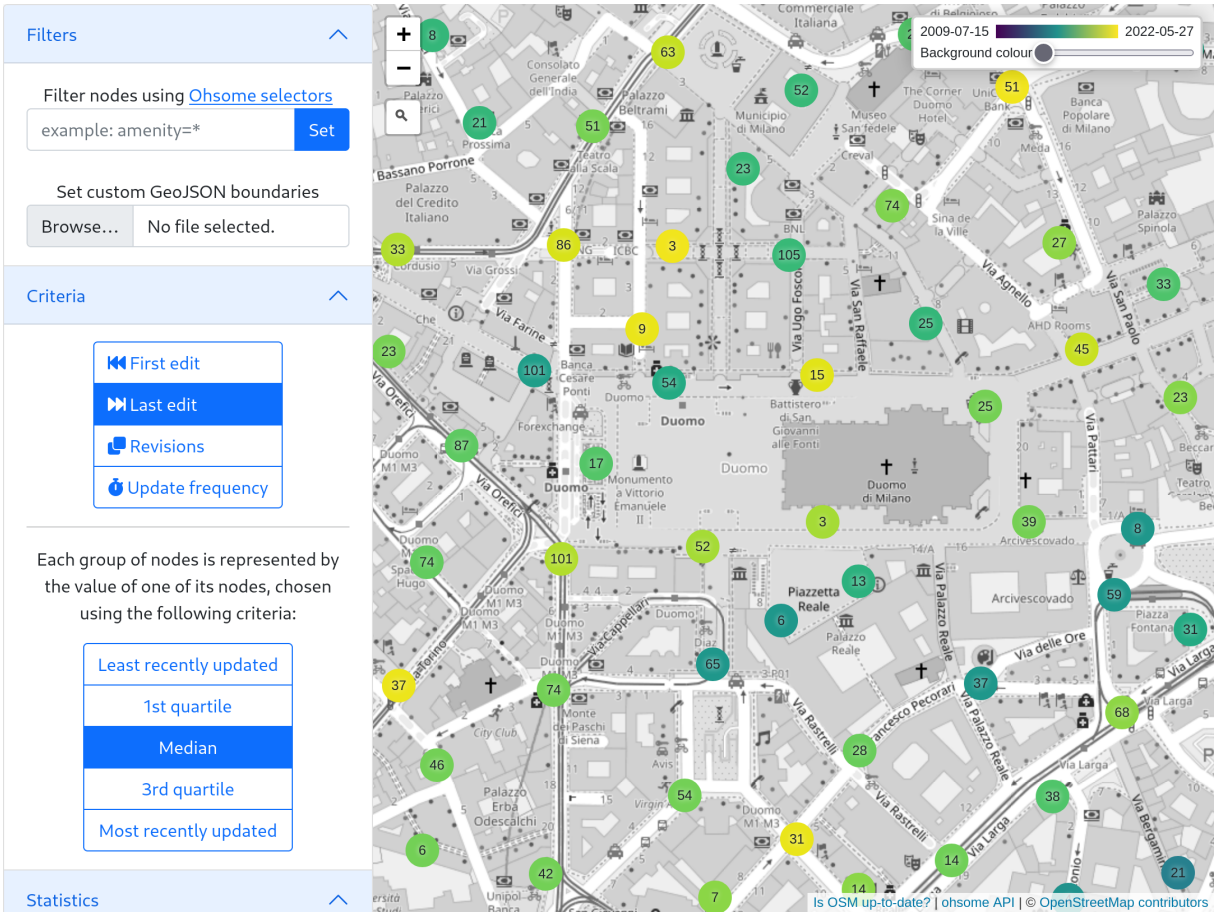


Figure 4.2: Web app – Zoom level 17, clustered nodes

Left bar

The left bar is composed by different section that can collapse (expanded by default). These are:

- **Filters**, used to reduce the number of nodes considered, using two mechanisms:
  - tag filters, that rely on Ohsome selectors, allowing the user to select only certain keys or values using different criteria;
  - GeoJSON boundaries, specified with a file input field, which allow the user uploading a set of geometries delimiting the area to be analysed (figure 4.4);
- **Criteria**, split in two sections:

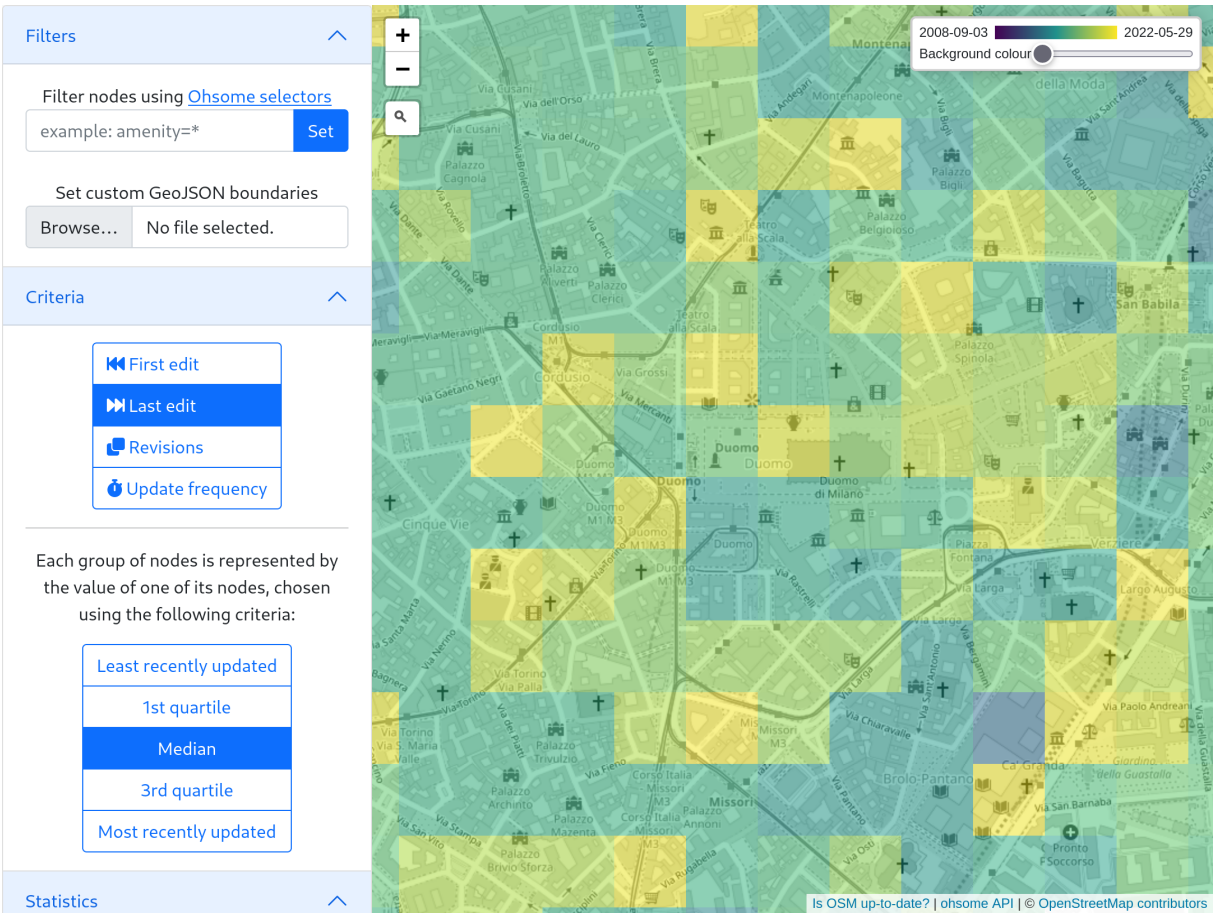


Figure 4.3: Web app – Zoom level 16, tiles

– **gparameter** to be evaluated, as previously described (see 3.2.1), with the exclusion of the user count (due to Ohsome limitations);

– **grouping criteria**, based on quartiles, which provides five choices:

  * minimum value, which is labelled differently based on the criteria that has been selected, so the minimum value button contains the text *Least recently updated* when *Last edit* is chosen as criteria;

  * 1st quartile;

  * median (default choice);

  * 3rd quartile;

  * maximum value, which is labelled using the same mechanism used for the minimum value, so the minimum value button contains the text *Most recently updated* when *Last edit* is chosen as criteria;

- **Statistics**, which list the values of the quartiles used by the grouping criteria, with the addition of the node count;
- **Save**, to download all the data, included the computed properties, as GeoJSON, and the statistics, as JSON file.

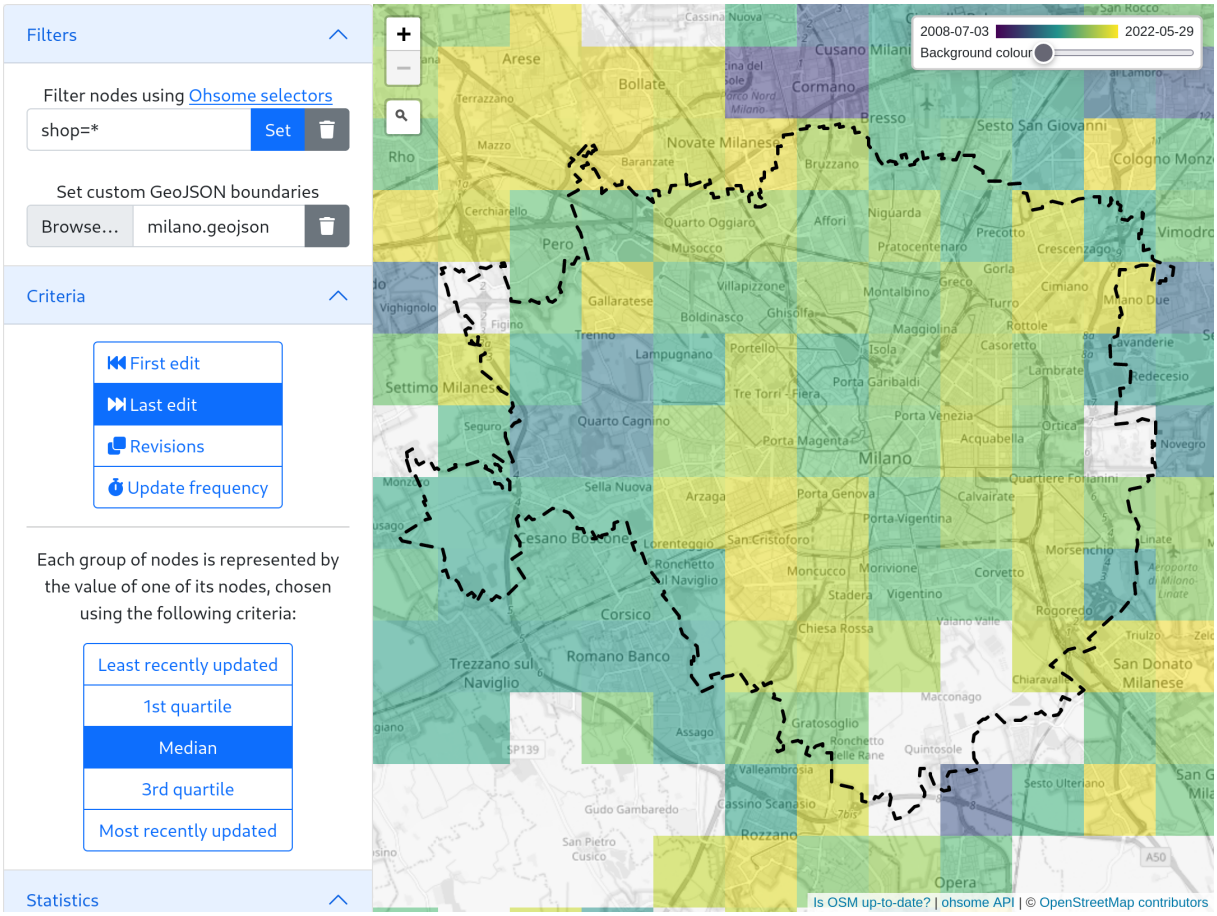


Figure 4.4: Web app – Shops within Milano administrative boundary

4.1.2. Command line

The software initially provided a command line interface (also called *CLI*), in addition to the HTTP API, by relying on the Hug library. As the focus shifted more on the web capabilities of the software and Hug was posing some technical limitations (such as the inability of streamed responses, which is useful for large data transfers), the CLI has been removed, but the user can still avoid using a browser by using software like the popular curl [32] or wget [182], or some more task-specific tool, like httpie [65].

Here is an example of data fetching within a bounding box using httpie as client:

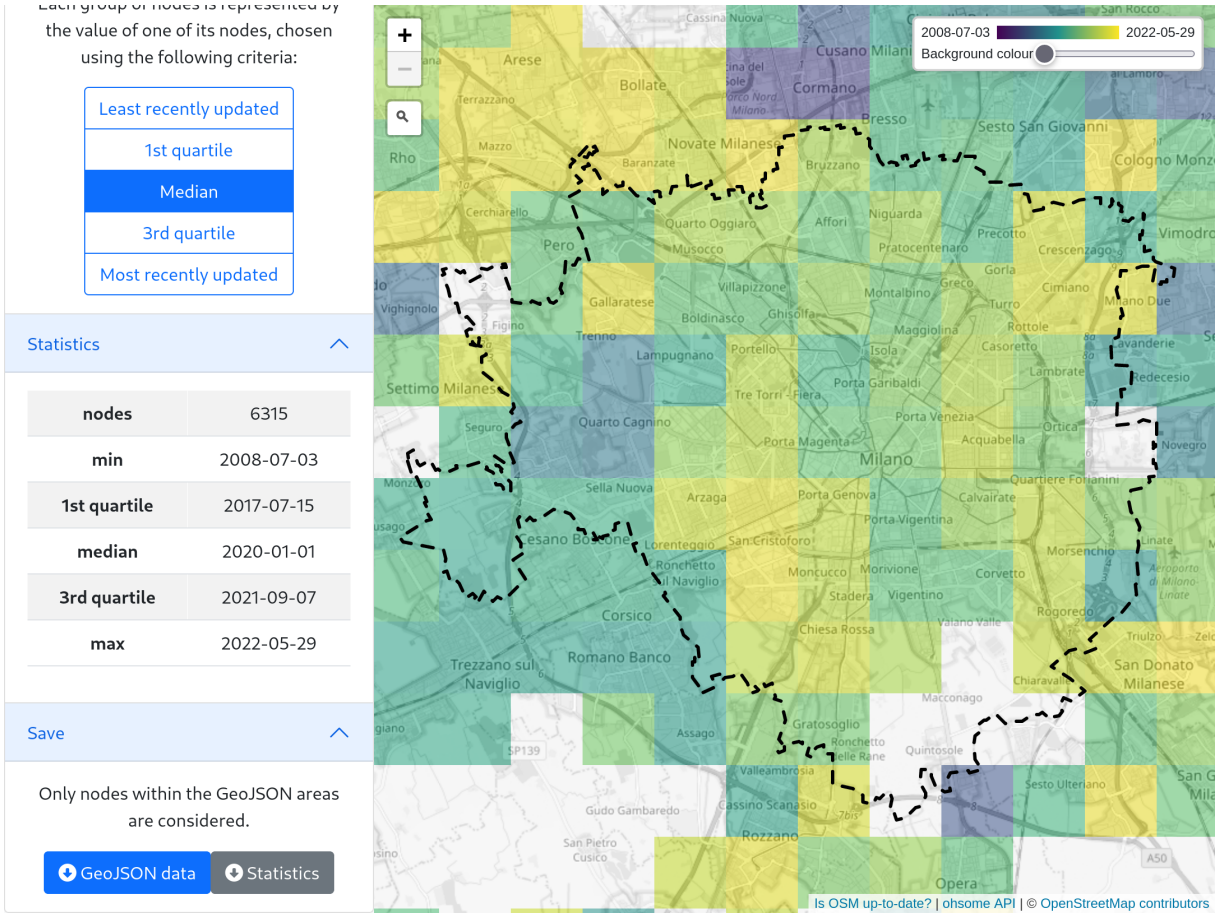


Figure 4.5: Web app – Shops within Milano administrative boundary, scrolled down to show Statistics and Save sections

```
$ https --download is-osm-uptodate.frafra.eu/api/getData \
    minx==9.2249536 miny==45.4767138 \
    maxx==9.2301893 maxy==45.4796778
```

Expected output (some values might differ):

```
HTTP/1.1 200 OK
content-disposition: attachment; filename="is-osm-
    uptodate_20071008T00_20220619T20.geojson"
content-type: application/json
date: Sun, 26 Jun 2022 10:54:34 GMT
fly-cache-status: MISS
fly-request-id: 01G6FTH4P52H1H51C7PTBK263Y-fra
server: Fly/9ece5bcd (2022-06-21)
transfer-encoding: chunked
via: 1.1 fly.io
```

```
Downloading to is-osm-uptodate_20071008T00_20220619T20.geojson
Done. 228.6 kB in 00:1.32045 (173.1 kB/s)
```

Here is an example of statistics computation within a bounding box:

```
$ https --download is-osm-uptodate.frafra.eu/api/getStats \
    minx==9.2249536 miny==45.4767138 \
    maxx==9.2301893 maxy==45.4796778
```

The produced file is listed below (some values might differ):


```
{
    "creation": {
        "1st quartile": 1395486054.5,
        "3rd quartile": 1646904811.0,
        "max": 1655371562.0,
        "median": 1508604198.0,
        "min": 1191801600.0,
        "nodes": 1007
    },
    "frequency": {
        "1st quartile": 0.3442313737818296,
        "3rd quartile": 3.4433962264150946,
        "max": 73.0,
        "median": 0.4955872369314324,
        "min": 0.0720063128822253,
        "nodes": 1007
    },
    "lastedit": {
        "1st quartile": 1519998699.0,
        "3rd quartile": 1646904811.0,
        "max": 1655451567.0,
        "median": 1646904811.0,
        "min": 1218060291.0,
        "nodes": 1007
    },
    "revisions": {
        "1st quartile": 1.0,
```

```
28         "3rd quartile": 3.0,
29         "max": 22,
30         "median": 2.0,
31         "min": 1,
32         "nodes": 1007
33     }
34 }
```

4.1.3. GIS software

GeoJSON is a widely used format, supported by GDAL [52], which is used by the popular open-source GIS called QGIS [149], which supports GeoJSON. That makes it possible to open the data downloaded from the web interface, easy available on QGIS for further analysis.

Tiles can be used in QGIS too, starting from version 2.2. Such release includes safe zoom limits to prevent the server from being overloaded, a mechanism to upscale the resolution of the tiles (since QGIS interpolates the default low-resolution tiles to make them less grainy, which is an undesired mechanism in this scenario), and a brief documentation in the README file as well.

QGIS users should define a new XYZ Tile layer, with a minimum layer set to 11 and a maxim layer set to 19, and set the following URL:

`https://is-osm-uptodate.frafra.eu/tiles/{z}/{x}/{y}.png?upscale=512`

The *upscale* parameter is used to set the resolution of each tile, but it does not affect how many blocks the tile is divided into. This value is hardcoded, and is set to $8 \times 8 = 64$.

Other parameter that can be set are:

- **mode**, the criteria used for the classification (default: *lastedit*);
- **filter**, the Ohsome selector used to analyse some keys or values;
- **percentile**, the percentile which should be shown when grouping nodes (default: 50; range $[0; 100]$);
- **scalemin**, the value associated to the coldest colour (optional);
- **scalemax**, the value associated to the warmest colour (optional).

Every parameter is optional, including *scalemin* and *scalemax*, which default values are

set by the server to some reasonable defaults. The *getStats* function can be used to better tune these values.

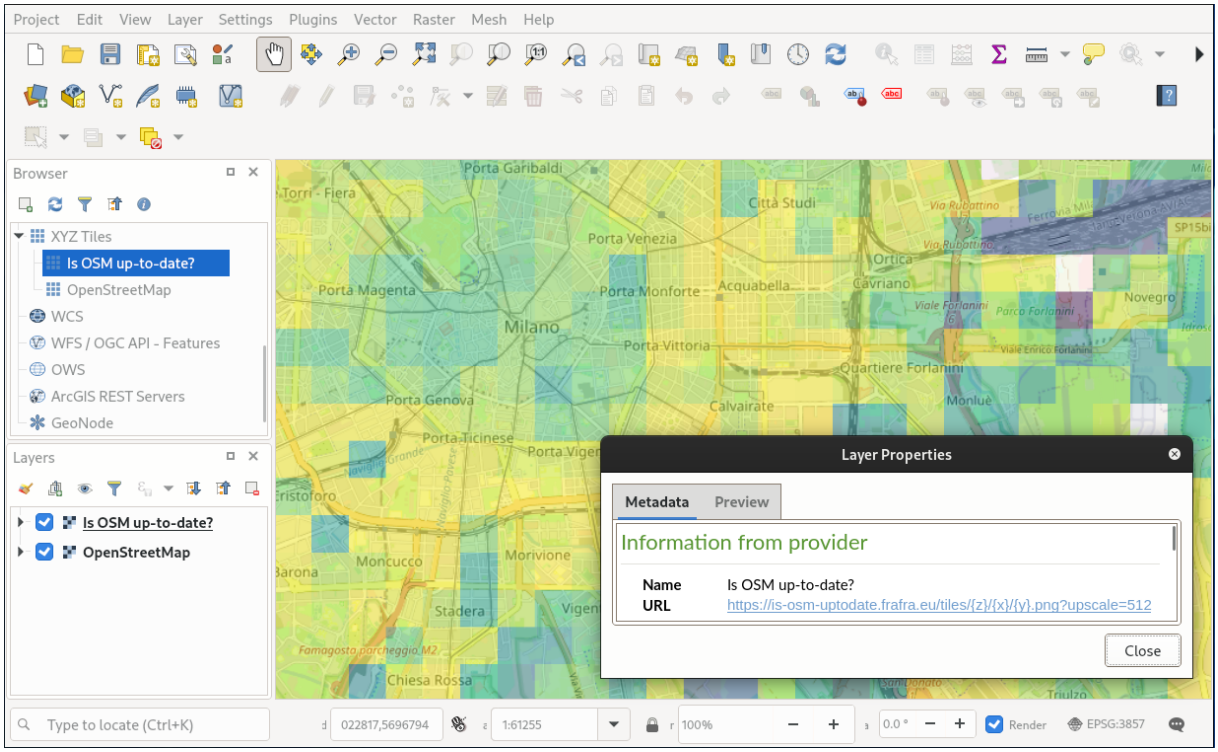


Figure 4.6: QGIS showing *Is OSM up-to-date?* tile layer

4.2. Code

The software consists of approximately 50 files and 2300 lines of code[1], underwent 370 revisions[2], and 13 releases[3]. ≈ 8000 lines have been added and ≈ 5000 have been deleted since the first prototype[4], excluding automatically generated files (such as lock files to pinpoint dependencies).

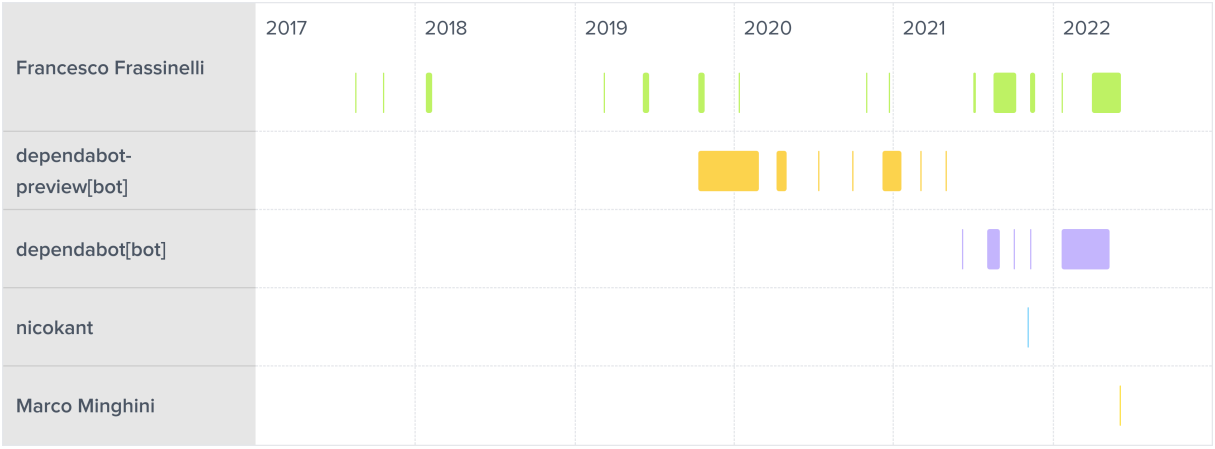
The code has been released under the AGPLv3 license [54], which is a copyleft license endorsed by the Free Software Foundation. AGPL licenses impose, in addition to the requirements of the popular GNU GPL license) to keep the software and its modifications open even when such software is not shipped as binary, but provided as a service.

[1] `cloc --vcs=git --not-match-f '[.-]lock'`

[2] `git rev-list --count v2.2.0`

[3] `git tag | grep ^v | wc -l`

[4] `git log --pretty=tformat: --numstat -- ':!:*[-.]lock*' | awk '{a+=$1; d+=$2} END {print a, d}'`



Graph 4.1: Development timeline generated using *Preceden* [72]

files	language	blank	comment	code
9	JavaScript	74	24	1039
14	Python	100	6	575
5	YAML	13	16	206
2	Markdown	55	0	107
3	CSS	15	0	105
3	TOML	24	3	97
3	JSON	0	0	91
3	Dockerfile	8	0	58
1	HTML	0	0	13
43	SUM	289	49	2291

Table 4.1: Lines of code, by language and type

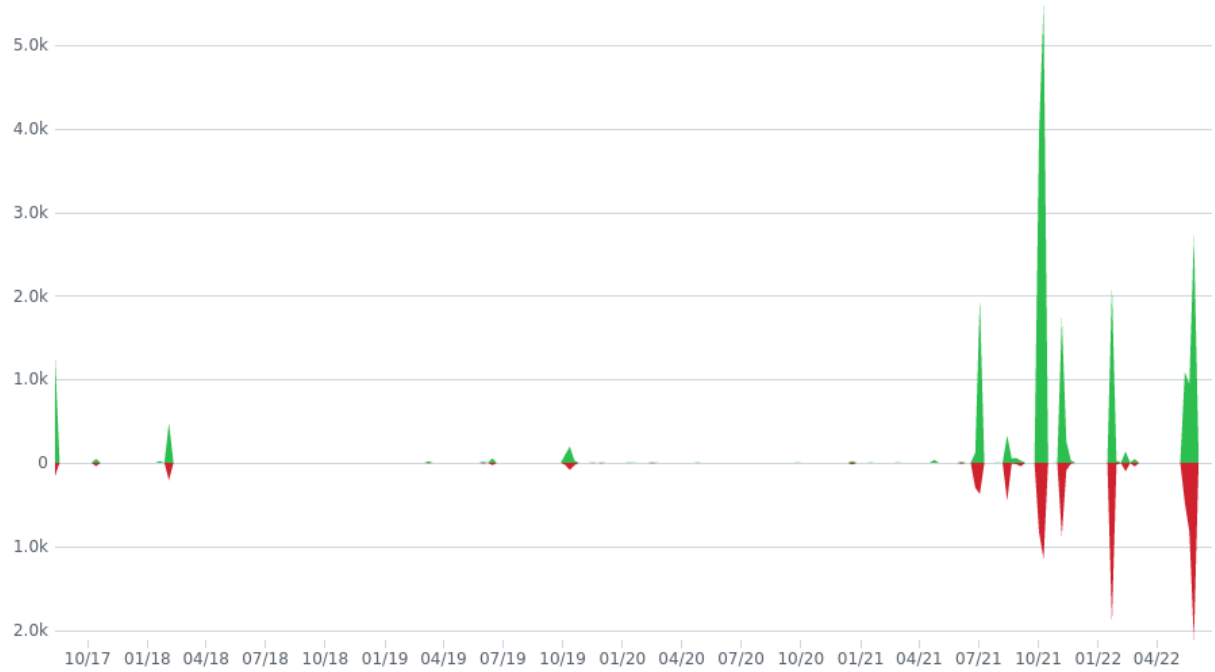
4.2.1. GitHub repository

The source code of such software is made available as a Git repository hosted on GitHub [50]. Two-thirds of the commits made by the main developer, Francesco Frassinelli, almost a third created by bots proposing version upgrades and triggering pipeline tests which have been manually verified and merged, and just one single commit by an external developer which extended the documentation.

The development is followed by 3 GitHub users, and the repository has been starred by 16 users.

4.2.2. Code quality

pre-commit [141] runs code formatting tools and checks before a commit is generated, to improve the overall code quality and consistency. In the case of *Is OSM up-to-date?*, pre-



Graph 4.2: Line added and removed, by week [27]

commit is configured to rely on black [144] (the most known Python code formatter), isort [146] (which rearranges import statements) and flake8 [145] to enforce a consistent code style and respect of common Python guidelines. eslint [40] is used to maintain consistency across the JavaScript assets.

In addition to that, LGTM [85] is executed automatically each time new code is linked to a pull request or pushed to the repository, looking for potential code vulnerabilities and errors.

Tests are defined using *pytest* [148] and *seleniumbase* [161] (which relies on Selenium [160]), to spot major regressions, both from CircleCI [28] is used to run a pipeline executing the tests and report the results by storing logs and automatic browser screenshots as artefacts.

4.2.3. Dependency management

A proper dependency management is crucial in today software development, even in simple codebases, as a single web applications can have hundreds if not thousands of dependencies, if indirect dependencies are considered. For that reason, it is crucial to pinpoint all the dependencies with their version, to improve reproducibility and avoid regressions.

Dependencies for the frontend are reported using the common solution of *package.json* for ranged dependencies and *package-lock.json* for pinned dependencies, using npm [103] as package manager.

Dependencies for the backend are specified using the newer *pyproject.toml* [136] and *pdm.lock* instead of the more common *requirements.txt*. *pdm* [133] is a relatively new dependency manager for Python, which adopted some recently approved Python Enhancement Proposals (PEPs) [136, 134] and even some that are still under evaluation [135]. One of the advantages of pdm over other mechanisms (such as *Poetry* [140] or *pip-compile* [76]) is its speed and the ability to take into consideration the required Python version when solving the dependencies.

In order to keep the dependencies up-to-date, the service *depend-a-bot* [33] has been used since the end of 2019, just after it has been acquired by GitHub in 2019, when it was still in beta. Such service opens pull requests automatically when a new version of a library or software referenced in the repository is released. A typical pull request contains an update of the pinned version of the software, as well as a recap of the most relevant changes. This mechanism triggers the test and build pipelines, and the results of these procedures are then reported in the pull request, giving a useful and immediate insight on the potential compatibility of the proposed update.

4.2.4. Docker

Container technology[5] has been adopted early in the project, as it greatly simplify reproducibility, deployment and development of heterogeneous and complex environments, with Docker [63] being the most popular solution to run containers. *docker compose* [37], a plugin of the CLI tool *docker* [36], is used to define different profiles, which cover the entire application lifecycle: development, testing and production. Being able to run containerized tests locally reduce the load on the continuous integration servers, and it is quicker, usually. Having a containerized development environment is greatly beneficial to use different Python versions and to have consistent results between development and production. The development environment is also configured with automatic reload trigger when the code change, so there is usually no need to rebuild the containers while making changes to it. Tests are defined in a different Dockerfile, so that the test container can be executed against different containers and servers.

[5]Containers are a popular method to bundle a generic application together with its dependencies, which is meant to be run in a sandboxed ephemeral environment

Good practices, such as multi-stage builds, compose profiles, YAML anchors, and per-directory caching (powered by BuildKit [22]) are used, defining a complete environment while being concise and keeping the build times and disk usage at minimum.

4.2.5. Monitoring

Sentry [13] is used to detect and collect errors and slow operations on both the web app and the server when the software is deployed in production. This helped to detect some corner case conditions and fix them. Such automatic mechanism is greatly beneficial, since the users are not often capable of producing high-quality bug reports with reproducible conditions, or they do not even notice minor bugs or glitches.

4.3. Performances

Four different areas have been chosen to measure the API endpoints perform in different scenarios. The software has been tested on a test server, which has the same technical characteristics as the one deployed in the production environment. The test environment consists of two virtual private servers on fly.io running Docker images on Firecracker. Each of them has 256 MB of RAM and a shared virtual CPU; the first hosts the main application, while the latter runs Redis [155], which act as cache. Both instances run in the same datacenter, in Frankfurt, Germany. Such location is the closest to the Ohsome servers currently available[6].

The functionalities that have been tested and timed are:

- **statistics** are going to be read from the */api/getStats* endpoint: it consists in a small JSON file which is transmitted after computing some basic statistics, which require the server to allocate enough memory to store all the features at once;
- **GeoJSON** is going to be downloaded from the */api/getData* endpoint while being processed: since it contains all the features, performances can be reduced by the client network bandwidth;
- **tiles** are going to be generated from the */tiles/{z}/{x}/{y}* endpoint: many requests are sent in parallel and the features, which can produce a higher load on the server, but the data being sent is actually very small: an 8 by 8 pixel PNG image.

Each endpoint has been tested with an empty cache and with an already populated cache,

[6] Ohsome servers are located in the area around Heidelberg and Stuttgart (Germany), within a radius of 200 KM from Frankfurt

and each measurement is executed three times. The mean value is displayed, as well as the minimum and the maximum, both rendered in the graphs using an error bar.

Tiles have been generated with a zoom level $z = 12$, which is the same size used by the caching mechanism. The criteria used for the tiles is *lastedit* (the default). Each tile render all the data included in its bounding box.

The software used to perform the benchmark has been published in the **benchmark** branch of the repository.

4.3.1. Piazza del Duomo, Milano

The first area is *Piazza del Duomo*, in Milan. The OSM feature representing it[7] covers an area of 45 163 m^2 and contains just 152 nodes with tags.

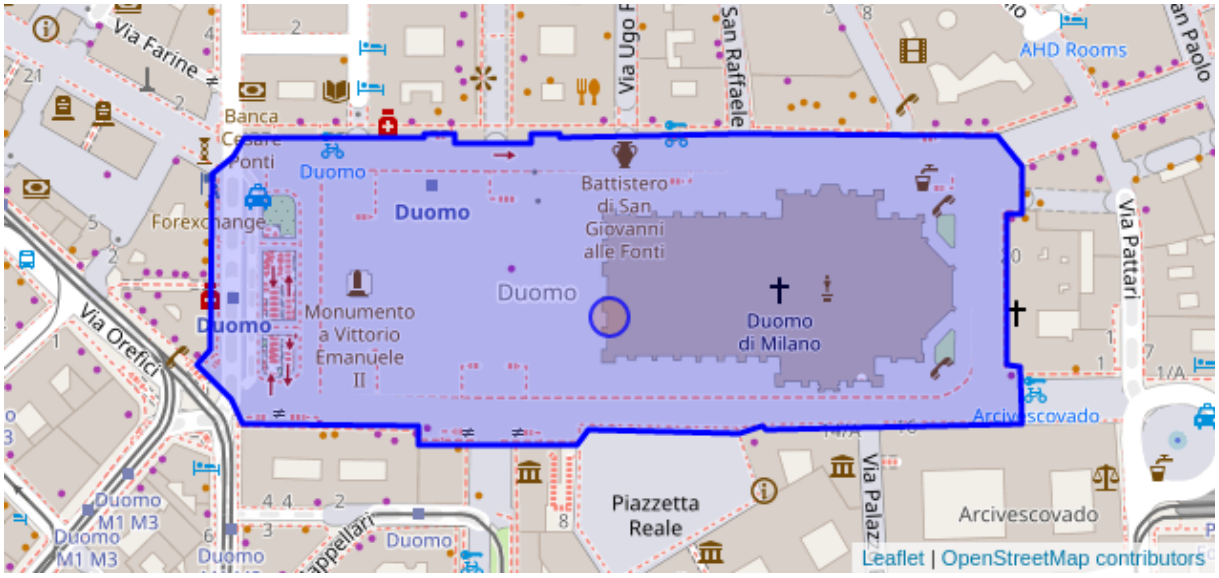
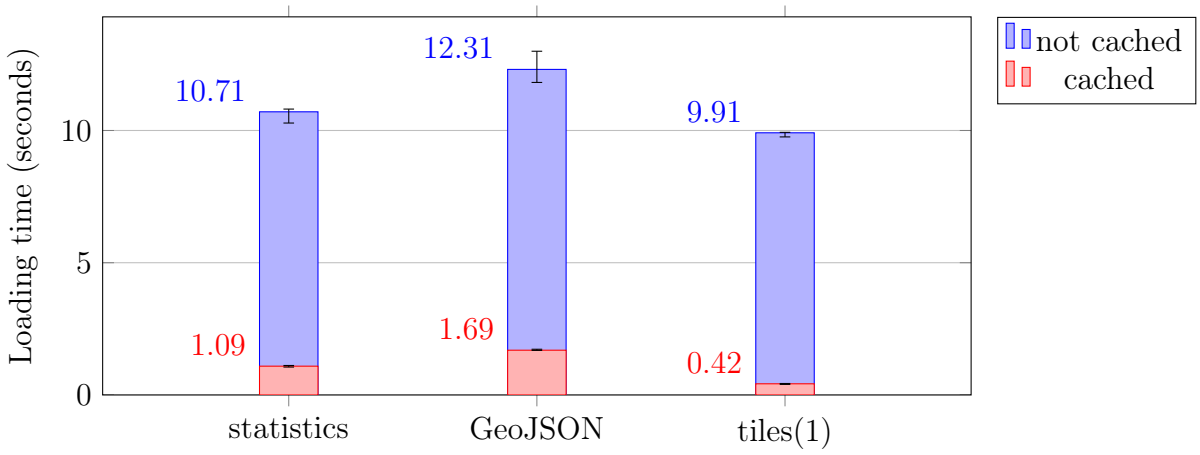


Figure 4.7: *Piazza del Duomo, Milano* shown by Nominatim

[7] OSM way ID 463529462

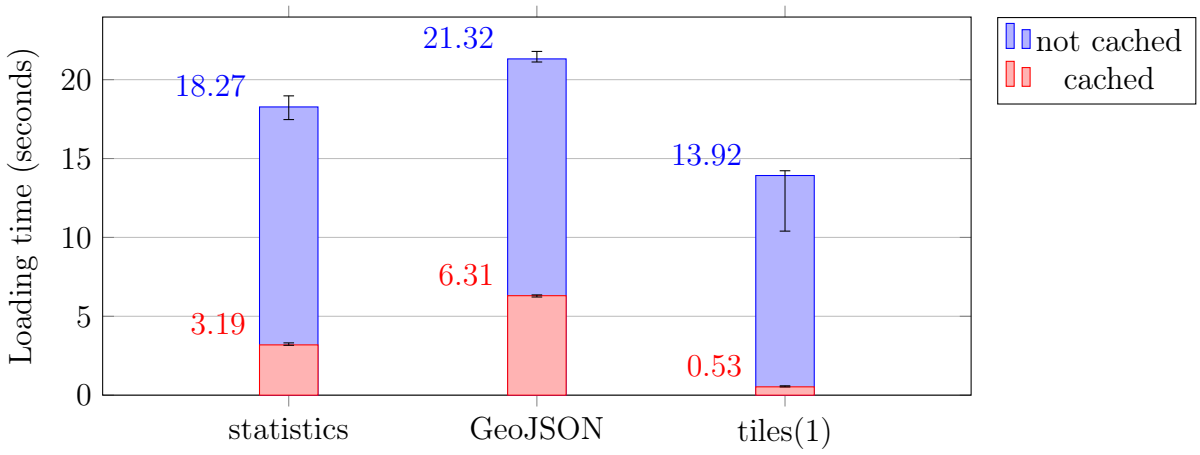


Graph 4.3: Software performances for *Piazza del Duomo, Milano*

As it can be seen, there is no such of a difference between the various endpoints. The software is 7 to 8 times slower when the cache is empty, when computing the statistics or returning the nodes as a GeoJSON. This is because the area requested to Ohsome is way bigger than the one requested, to get better performance in the future requests, as it is more efficient to request a bigger area than requesting subdivision of such area.

4.3.2. Municipio 1, Milano

Municipio 1 is a subdivision of Milan, which can be considered the historical city centre. The OSM feature representing it[8] covers an area of 9.4 km^2 and contains 27547 nodes with tags.



Graph 4.4: Software performances for *Municipio 1, Milano*

[8]OSM relation ID 3952986

Figure 4.8: *Municipio 1, Milano* shown by Nominatim

4.3.3. Milano

The OSM feature representing the *city of Milan*[9] covers an area of 182 km^2 and contains 149283 nodes with tags.

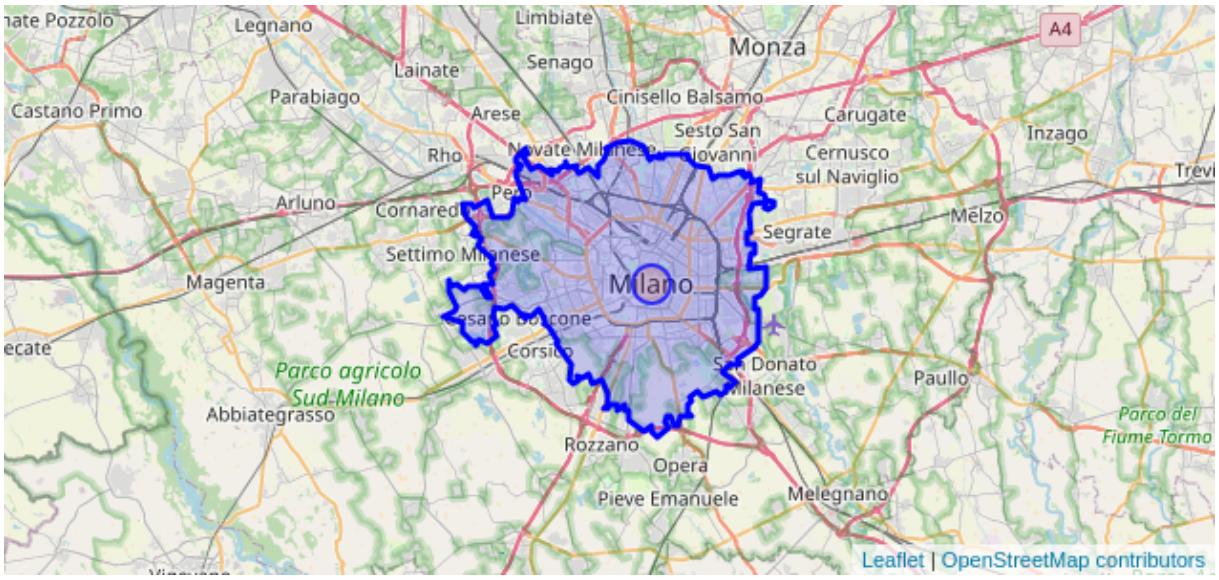
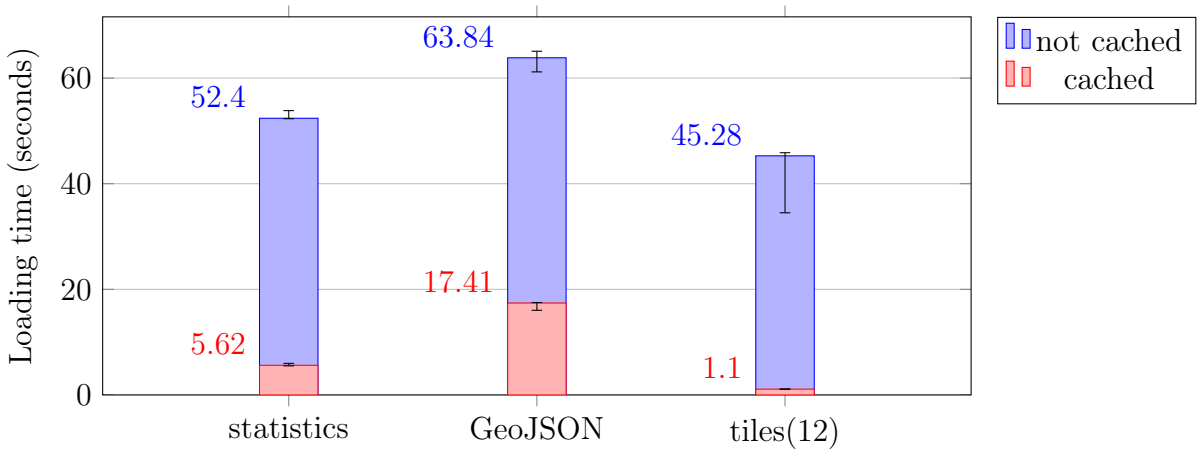


Figure 4.9: *Milano*, shown by Nominatim

[9]OSM relation ID 44915



Graph 4.5: Software performances for *Milano* city

4.3.4. Città Metropolitana di Milano

The OSM feature representing *Città Metropolitana di Milano*[10], which is an administrative boundary of level 6 in OSM, covers an area of 1577 km^2 and contains 334418 nodes with tags.

Such entity has an exclave. As a result, the boundary consists in a multipolygon, which is handled as well by the application. A quite unique peculiarity of such exclave is that it is quite far from the main polygon. The software can avoid downloading the features located between the two polygons, as they are not needed for the analysis.

During the test phase, it has been discovered that Fly.io terminates inactive connections which last more than 60 s, so this last test has been executed locally. It is still comparable to the previous tests, as differences between running the software locally or on a server have been found to be negligible for this comparison, showing that the main bottleneck is the data retrieval from Ohsome.

[10]OSM relation ID 44881

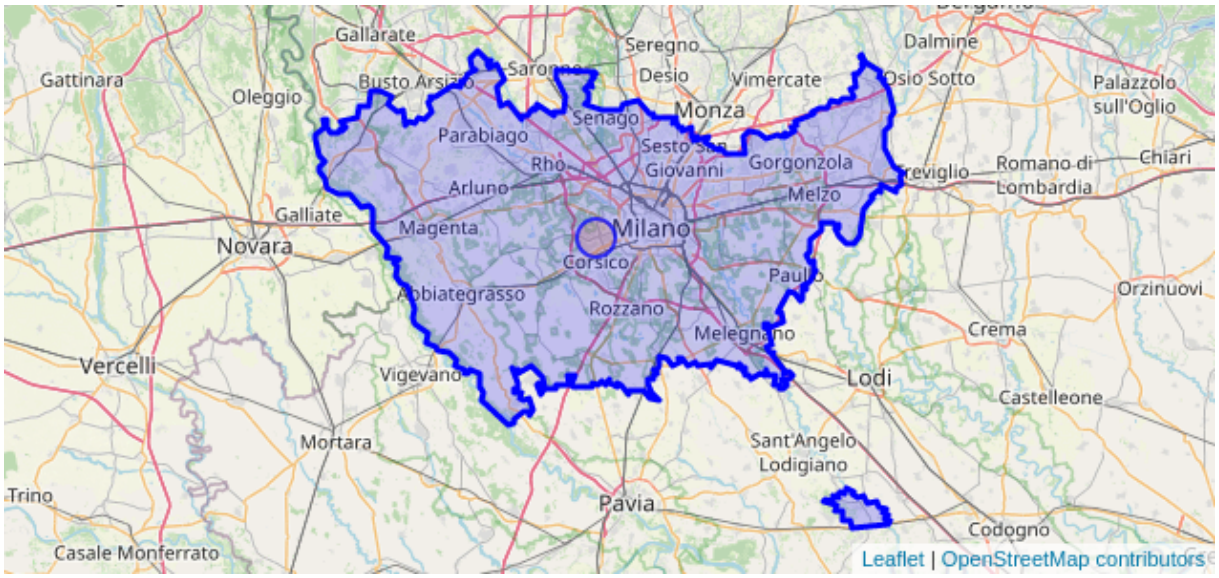
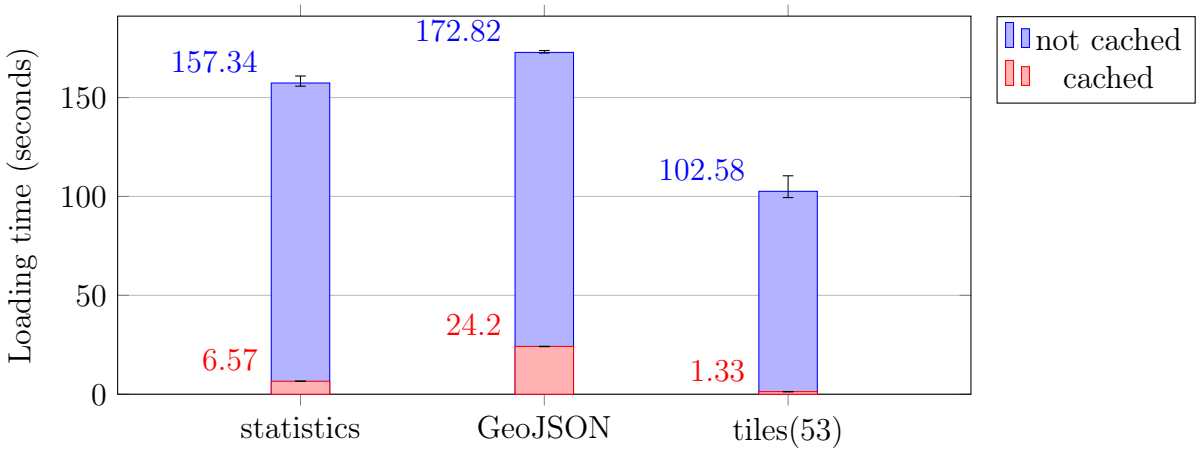


Figure 4.10: *Città Metropolitana di Milano*, shown by Nominatim

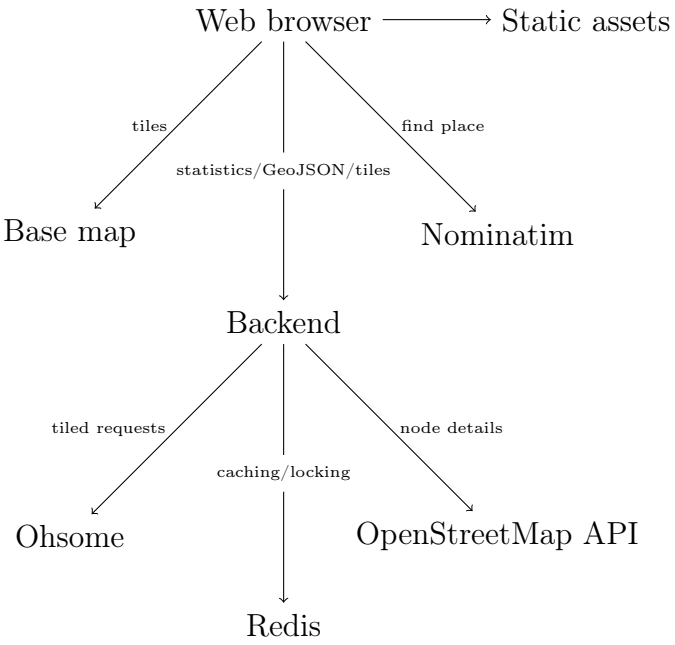


Graph 4.6: Software performances for *Città Metropolitana di Milano*

4.4. Architecture

The software consists of a web app, which sends requests to the backend, which relies on OpenStreetMap to provide details on specific nodes, and on Ohsome API to get historical data, which are then processed to produce statistics, tiles and GeoJSON output, where all nodes have some computed properties.

Requests to Ohsome are mediated by a caching mechanism powered by Redis, which is also used for locking, to coordinate requests to the Ohsome API across different threads, processes, or even instances.



Graph 4.7: Software architecture, conceptual map

4.4.1. Frontend

The web app has been initially written as a monolithic JavaScript file, using Leaflet [83] and Bootstrap [174] and Font Awesome [44] are used to produce a consistent and pleasant interface

The web application has been rewritten using the popular React [154] for version 1.8, and a complete UI redesign for version 1.9. Such choices have been made to handle the growing complexity of the interface, as well as and making it easier for other developers to approach the software in the future and contribute to it. *L.markercluster* [82], with the addition of the clustering extension adapted for React named *react-leaflet-markercluster* [176], which does not seem to be maintained at the moment, so some open bugs forced to create a forked version containing some fixes needed for *Is OSM up-to-date?*, which have been proposed to be merged back [74].

Webpack [180] is used to bundle and optimize the assets, as well as for the live-reloading capabilities used during the development.

4.4.2. Backend

The software adapted aiohttp [14] combined with gunicorn [58], which allowed to fetch and serve many tiles concurrently with a lower latency and resources.

To further reduce the memory usage, the software takes advantage of a streaming JSON parser called jsonslicer [89], which relies on the fast C JSON parser YAJL [62], as well as some memory-saving Python techniques as the usage of the yield statement whenever possible.

The mercantile [90] library is used to compute tiles and the bounding boxes for cached data, while pygeos and shapely have been added in version 2.2 to handle arbitrary GeoJSON boundaries.

GeoJSON boundaries are parsed and applied using Shapely [162] and PyGEOS [147].

4.4.3. Caching

Caching the response to arbitrary bounding box requests makes caching useless, as each of the requests can have a different bounding box, even if slightly. To address this issue, each bounding box is normalized into a set of one or more bounding boxes belonging to a known grid, which is defined as a standard regular tile grid at a specific zoom level. Each of the resulting bounding box tiles are encoded as quadkey [153], which is an efficient method to encode a tile using a single number. Quadkeys has also been chosen as fetching a tile composed by many can be done by requesting all the tiles that share as prefix the quadkey of the wider tile[11].

The software then checks if the desired areas as already been cached by looking at the Redis database. If the data is not there, a lock is created on the tile, to signal that the tile is in the process of being requested to the Ohsome API, so other requests of the same tiles are going to wait, and duplicate requests are avoided; the software then stores the results and removes the lock so that all requests can be satisfied. If the data is in the cache, there is no need to fetch data or to use locks. The access to Redis is mediated by walrus [84], which provides some useful abstraction and safety, like its locking mechanism.

Each tile has a TTL (Time-To-Live) of a month, before it gets discarded automatically. To prevent the cache from getting full, a soft-limit of 800 MB is enforced using a LFU (Least Frequently Used) policy, so that when such limit is reached, Redis stats to automatically drop the least popular tiles. This mechanism has proved to be very effective and flexible to different setups, as the software can easily take advantage of the available memory of the system.

Redis also has some spatial capabilities, but they are not considered to be useful, since

[11] For example, the data of the tile having 124 as quadkey can be computed by aggregating the data of tiles starting with 1; the list of these tiles can be obtained by executing `KEYS 124*` in Redis

no TTL or cache eviction policy can be specified on the members of a geometry set.

4.5. Limitations

4.5.1. Area of study

The web app is locked on a minimum zoom level of 12 due to performance limitations, which means that the tiles are loaded with $z = 11$; such limit is also enforced for requests coming from other clients (such as GIS software). It is possible to use the *getData* for wider areas, thanks to its streaming capabilities, but it has not been extensively tested. *getStats* could take too much time to start giving a reply (since it needs to load all the data first) and the connection could be terminated before the computation ends, while using too much memory on the server.

4.5.2. Relation to GIS software

The web app is not a full-fledged GIS software, nor it aims to be one. Its features are limited to the scope of evaluating OSM uptodateness. There is also the risk of wasting time and resources by trying to reimplement features already available on popular GIS solutions. This is the reason GeoJSON and web tiles has been used: to provide standard interfaces that could be used by external tools. The integration with GIS software is limited and could be greatly improved.

5 | Reception from the community

A survey targeting version 2.2 [169] has been shared with people involved with OpenStreetMap (both mapping and research activities), to evaluate how the software is perceived and if it is considered useful. It has been asked to test the software first and fill the form afterwards. The survey has been made available in both English and Italian. Instead of sharing a direct link to the survey, a dedicated wiki page on the GitHub repository has been created, which contains the links to the survey in both languages and their results.

The software used for the survey is *CryptPad* [31], an open-source web application focused on user privacy, with end-to-end encryption.

The survey has been shared within the following communities:

- @OpenStreetMapItalia group chat on Telegram;
- @Polimappers group chat on Telegram;
- science@openstreetmap.org mailing list;
- talk-it@openstreetma.org mailing list.

The survey has also been shared by some users on Twitter and by email.

All the answers provided in Italian have been translated to English for consistency.

5.1. Respondents

23 people filled the form, and they replied to the question "*Which user case fits you the most?*" (which allowed multiple answers) by describing themselves as OSM contributor in 16 cases, researchers or students in 5 cases and just one as user. 2 selected the option "*Other*".

5.2. Features

It has been asked to each respondent to give a score for each feature, ranging from *terrible* to *great*, using *bad*, *average*, and *good* as intermediate options. The question has been formulated as the following: "*How would you rate these features of is-osm-uptodate?*".

Two users replied with a different scale, using stars instead of text, from 1 (minimum) to 5 (maximum). It has been found that a user gave almost exclusively 1 and 2 stars, while still providing a positive final overall comment on the solution. It has been assumed that the question has been misunderstood, as the user could have it interpreted as a request of rating the features based on their importance, not on how bad or well they have been implemented, or the user could be not familiar with the paradigm of the stars as paradigm to express a vote. The scores given by such user have been ignored in this section, and the survey has been improved to avoid this potential confusion, by using textual labels.

Few other responses have been excluded, as a user wrote: "*My "terrible" answers above can be ignored. I did not try, so I needed a "no idea" option*". The software used for the survey allow seeing non-aggregated data, which made it possible to fix the results as requested.

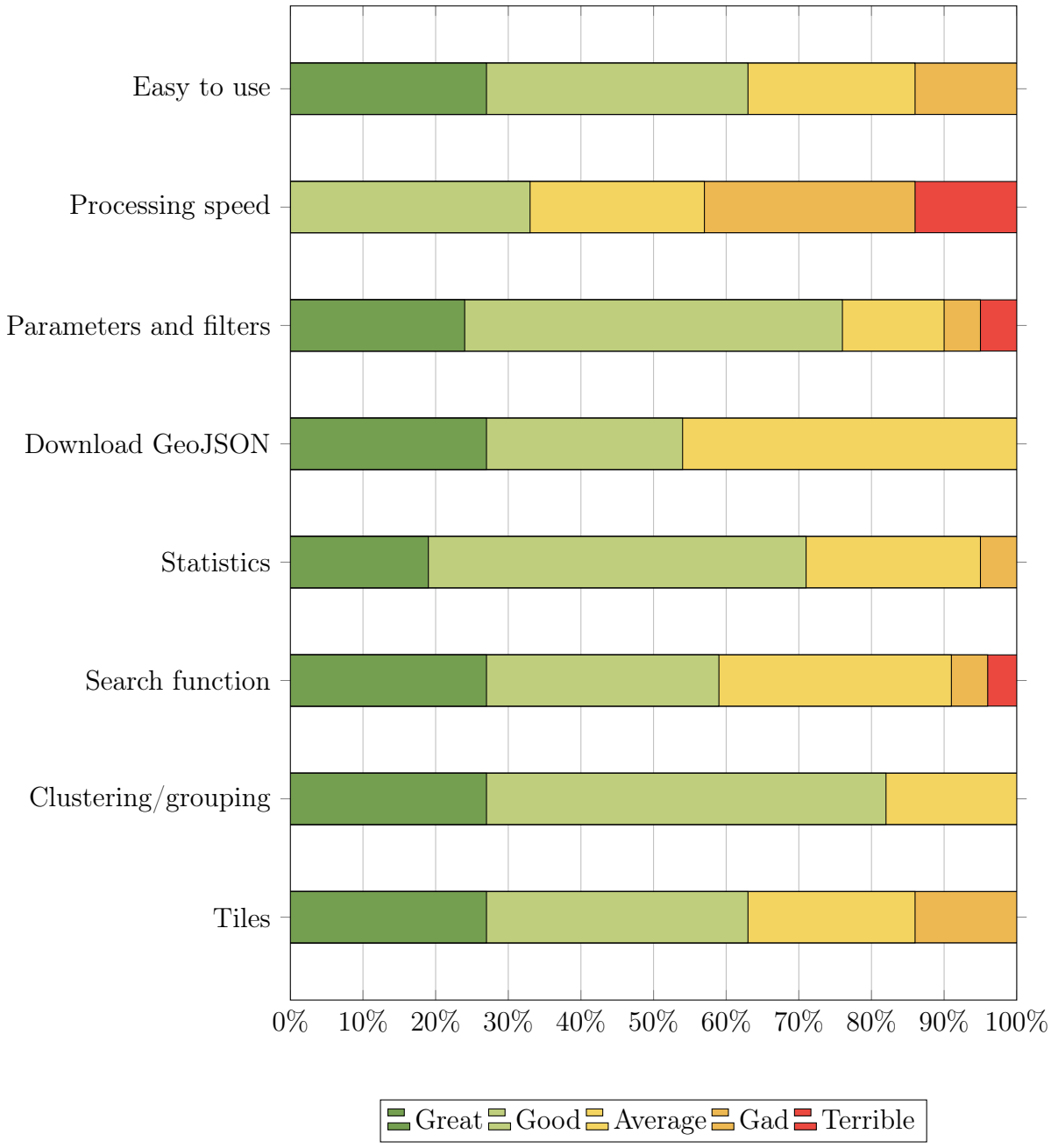
Easy to use

The software seems relatively easy to use for most of the user, which can be a good result for such a usual software, with some unique patterns of interactions, but there is still room for improvements.

Processing speed

The processing speed seems to be the area where the software got the worst scores. This could be due to different reasons:

1. Two *terrible* votes have been cast while the software was experiencing a technical issue: the Redis database has been unreachable by the main application, which has no retry mechanism, since it has not been taken into account connectivity issues. Such a scenario has never been seen before, as both services have been executed on the same machine, in an ordered manner. *fly.io*, which is the service that has been recently used to deploy the application, starts two virtual machines independently of their statuses or availability. A restart of the Redis database could cause a network failure. The mechanism to restart containers based on health checks failed too. The services have been restarted manually few days after.



Graph 5.1: Scores users gave to each feature

2. Some users tried to reach a location by dragging the map around repeatedly instead of using the search function, which results in a lot of calls to the server and higher waiting times. Previous versions of the software did not load the data on each boundary change, but they relied on a button to fetch the data. This is an effective method to avoid loading unnecessary data, but it reduces the usability of the platform. A potential solution could be to revert to the previous behaviour and adding a keyboard shortcut for users that needs to fetch data repeatedly, as well as some mechanism to avoid loading new tiles automatically.
3. Some users could have very high expectations on how quickly such a system could be able to fetch and process the data, even if the solution is based on Ohsome (which is the fastest public API for fetching historical OSM data) and it can be even noticeably faster than it when the data has been cached.

Parameters and filters

Parameters and filters seems relatively good, but the service disruption could have had an effect on this question too. Even when excluding the worst score, it looks like a user does not feel comfortable enough with the given filters. It could be due to not being familiar with the concept such quantiles, or by the syntax used by Ohsome to specify tag filters. A help text with a link has been added afterwards, but the interface could be further improved to assist the user in defining tags.

GeoJSON download

The mechanism to download the data as GeoJSON is rather simple: the user needs to click a button and the download starts immediately. This simple question further suggest that the outlier could be due to a temporary and technical fault of the solution which affected almost all of its functionalities.

Statistics

Statistics got a good score. They are consistent with the criteria, and they can be downloaded too, which seems what the users expected by a relatively simple, yet important, functionality.

Search function

The search functionality relies on a very common Leaflet widget, so the results were in line with the expectations.

One user gave the lowest score possible, but such user could not find the search button. Other users had the same problem, so the appearance of the search widget should be changed.

Clustering/grouping

The clustering/grouping feature received a good score, which seems to confirm the development direction taken during the last year. The *bad* vote could be due to suboptimal performances (some mobile devices can lag when using the clustering feature on many nodes) or to a not-so-intuitive subdivision of the areas containing the clustered nodes (such areas are visible only when hovering the mouse over a clustered node). These areas could be made visible all the time or a Vonoroi diagram (consistent with the clustering method) could be generated and shown as an additional layer, to better represent how the space is subdivided. The *terrible* vote could be linked to bad performances on some low powered devices, such as smartphones.

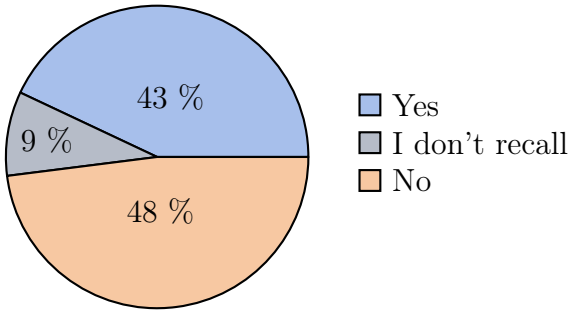
Tiles

What could be considered as the best result came with the last question, which shows that the recent addition of the tiles as a method to aggregate data has proven to be an effective and intuitive method. Tiles received less average and negative votes compared to the usual clustering.

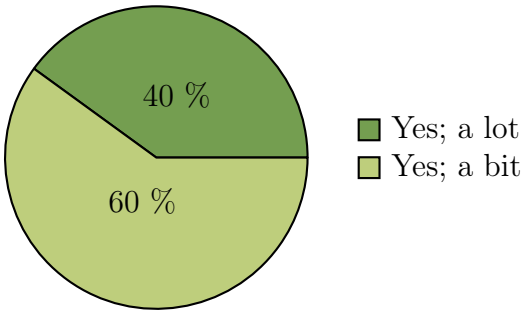
Such a final result suggests that regular tiling should be considered as the main (and potentially the only) method to group nodes.

5.3. General questions

Almost half of the respondents have used the software before. All of them agreed that the software improved recently.



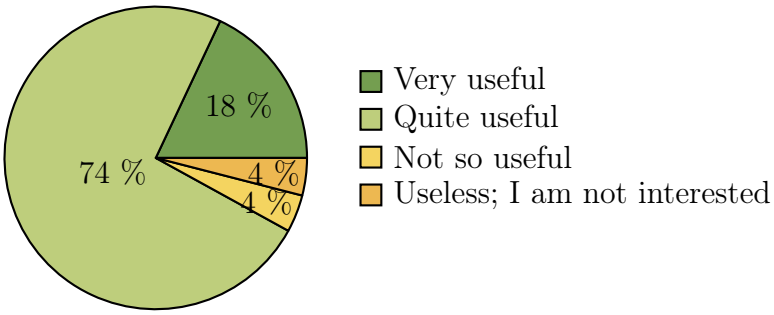
Graph 5.2: Survey – Have you tried it before?



Graph 5.3: Survey – Do you think it has been improved recently?

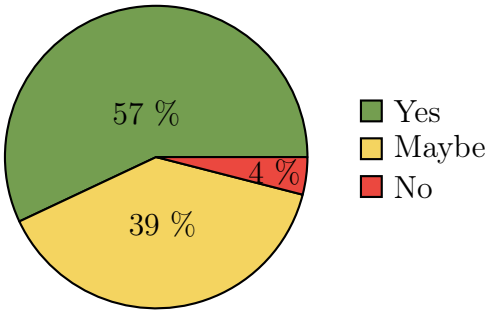
"*No; it is slightly worse*" or "*No; it got a lot worse*" have not been selected by anyone.

It has then been asked to the respondents if they like the software, if they intend using it in the future, and for which purpose.



Graph 5.4: Survey – Is it useful?

No one replied "*Useless; I'd rather rely on other tools or services*". Such result could be considered as a final score on the software, which is a positive one. One user is not interested in the software, but is not willing to consider other solutions, which could hint at the fact that such user is not the target of *Is OSM up-to-date*.



Graph 5.5: Survey – Would you use it in the future?

Respondents are almost equally split between the ones that are going to use the software again, and the ones that are not sure about that. The only negative result came from the user that is not interested at all in this kind of tools. Improving the most critical aspects that have been highlighted in the survey could increase the user retention and make the software more popular in the community.

5.4. Possible usages

The respondents would use the software for improving OpenStreetMap data in 13 cases (it is reasonable to assume that these users would do data gardening, mostly), and for doing research or analysis in 8 cases. 2 respondents also checked the option "*Other*", suggesting other ways to use the tool. Multiple choices were allowed.

Just 2 respondents that did not qualify themselves as researchers or students, replied to the question "*For which purposes are you going to use it?*" suggesting that the software could be interesting to explain "*[...] OSM data quality to potential users*" and to check "*[...] the contribution trend in my interest area*".

Almost all respondents that qualify themselves as researchers or students, replied to the question "*Which kind of research or analysis are you thinking to do with it?*".

5.4.1. Practical applications

A first group of answers have in common the focus on figuring out which areas need to be mapped or updated first:

- "*Combine with data from Disaster Ninja to get prioritize area to map.*"
- "*Research on prioritization of areas or types of elements in updating information in OSM. Connection between presence of users (and how many or expert/new) and frequency of updates.*"
- "*Automatic identification of areas where OSM might need updates*"

Another answer is still focused on practical applications, but more on monitoring instead of mapping: "*[...] I am thinking about a rapid data quality monitoring tool of critical infrastructures in global south, where the OSM contributions are intensive and quickly updated*".

Having researchers considering the tool in such scenarios is a valuable feedback, suggesting that the software could have a positive impact on various areas of the world, especially if

further improved.

A smaller group of answers is more focused on the research aspects:

- "*As an academic project to quantify various statements made on the use and validity of OSM*"
- "*Data quality analysis, extent of interaction with the data*"

One respondent, instead, is going to "*[use] this for student assignments.*".

These answers hint that the software could be used as starting point for future research projects regarding OpenStreetMap temporal accuracy and up-to-dateness.

5.5. Additional comments

Some respondents left a final comment, that could be worth discussing.

Comments are mostly exclusively positive, such as "*great tool!*", "*Great work, congratulations!*" and similar, or more articulated ones, like "*Definitely a nice attempt to explore OSM intrinsic data indicators*", or "*this is an interesting tool and I think it can be useful for both mappers and data users!*".

Various users noticed technical issues, glitches or difficulties in using the interface. For example, some of them failed to spot the search icon to find and move to the desired location, or they triggered the loading of so many data during their movements on the map that the load times grew too much. Most of the discovered limitations were known already, but it has been valuable to know which ones have been hit the users the most and how.

Some improvements have been suggested, such as:

- remove the restriction on the minimum zoom level while avoiding the tile loading on even wider areas: that would allow users to roam freely around the globe, without having to rely on the search function;
- store the last location in a cookie, instead of the URL, so that the user would always land on the last visited location;
- allow setting a different colour range;
- add information about the software and how to use it within the application itself;
- show tag filters in the legend;

- hide the grouping criteria when data is not grouped;
- add tags data to the downloaded file;
- analyse ways too.

Just the last two suggestions would require a major change in how the software works, since more data would need to be fetched and the system would become too slow. This feature would then need to found an even better mechanism than fetching data from Ohsome APIs.

A discussion started on the Italian OSM mailing list called *talk-it* [48]. 5 users wrote different comments and consideration that were not in the result of the survey. Reaction were mixed: some users are puzzled by the software and its goals, and they would probably benefit from having a global quality index related to temporal accuracy, while others were very happy about the tools, or interested, and found it useful. One of them also mentioned the *check_date* tag [81], which can be used to specify when a feature has been checked, thus having a better understanding if a node hasn't been changed for a while because there was no need to update it or if it has been neglected. Such tag is not widely used [25], and in case contributors start to use it more often, the software would still register a change of such tag as a new revision, thus improving its metric without any further change.

6 | Conclusions

6.1. Final considerations

Is OSM-up-to-date? proposes a new solution to analyse OSM history, providing to both contributors and researchers an additional tool with unique features to explore the data and perform their tasks, thus moving forward the estimation of the temporal intrinsic quality of OSM data.

The development of the software posed major difficulties, especially when moving from a working prototype to a production solution that could work efficiently on areas that are 3 orders of magnitude bigger, requiring to change the software stack multiple times, evaluating different approaches. However, the work proved that it is possible to run historical analysis on OSM data using modest resources within minutes across hundreds of thousands of nodes. Working on it and exploring different technical methods provided new insights on how this class of problems can be tackled.

The presentation of the intermediate results at different conferences and to the OSM communities resulted in interesting feedback and opportunities for improvement. Writing papers on the matter required to have a better understanding about the state of the art on the topic and brought the research to adhere to higher quality standards.

The feedback from the community has been mostly positive. In addition to the many OSM contributors interested in data gardening who found the tool interesting, there is an interest in the academic environment to use the software for scientific purposes. The technical evaluations, benchmarks, and trials made during its development could be considered a starting point for implementing more sophisticated solutions that could be even more appealing for a research-focused audience. The hope is that the research in this field can further progress and quality indexes for intrinsic temporal accuracy can be defined so that they can be implemented in *Is OSM up-to-date?* and other software.

As a final result of this research and development process, a list of future work and research activities that could be done is shown in the following section.

6.2. Future work

6.2.1. Research topics

Overall quality index

This is probably the most important goal that is left from a research perspective: finding a formula or a method that could provide a quantitative estimation on the temporal accuracy of the OSM data, that goes beyond the computation of the average update frequency, last edit or number of revisions, capable of taking into account tag classes and calibrating the result based on spatial trends.

6.2.2. Features

Improved usability

Usability would greatly benefit from the changes proposed by the respondents to the survey (see 5.5). Users' feedback demonstrated that it is crucial to find a different way to let the user search for a specific location. A horizontal bar to type the location could be used, but it would clash with the legend in the top-right corner on smaller screens; a set of CSS rules could be made to avoid such an issue.

Other improvements that have been proposed are linked to the map handling as well, like giving the user the freedom to zoom out as much as needed, without any limit, or store the last visited location.

Custom boundary for tiles

It is not possible to specify a custom GeoJSON boundary for tiles, which are always rendered without applying any spatial filter on the tags. It could be useful to crop data based on the boundary before computing the tile, but users could be puzzled when seeing non-transparent pixels placed over a boundary line: how would it be understood? The user might ask if the software has taken into consideration features just across the boundary as well. This problem has been solved when producing hexagonal maps on QGIS by cropping the hexagons using the boundaries, which seems the optimal solution that should be implemented in the new software as well, but tiles are raster data, not vector, so a different API should be developed to produce vector tile-like data. Another solution would be to upscale the raster before cropping it to match the boundary so that it would be less noticeable to the user that the graphical representation of the tile does

entirely match with the boundary.

Export area as GeoTIFF

The software allows to download statistics and a GeoJSON representing all the features with their computed properties, but it does not allow to download a single GeoTIFF file of the area under analysis. While the tile layer allows having such a functionality outside of the software, by relying on specific tools or extensions to GIS software, providing to the user a single button would greatly improve the user experience.

Such improvement would greatly benefit from the possibility to use custom boundaries on tiles.

Better error handling

The server catches various exceptions, but the current user interface lacks the capability to display errors in a meaningful way to the user. The service Sentry (see 4.2.5) highlighted some rare errors as well, such as cache locking failures related to unexpected reboots or timeouts. Various errors are also triggered by the fact that the software relies on external services, which are not always available; users could attribute the cause of the issue they might experience to the software instead of the external API in some cases.

Better GIS integration

Another potential improvement would be to create a plugin to improve the integration between some popular GIS software and the application, by calibrating the raster scale based on statistics, for example, or by developing a user-friendly interface to configure the various parameters, or by providing some good presets or demo project that can be imported and used.

6.2.3. Performances

Approximate statistics

One of the main limitations of the *getStats* endpoint is that the whole result is sent at the end, and the server temporarily stores all the features properties in memory at once, to compute that. A possible trade-off between accuracy and speed could come by computing approximated statistics while processing data as a stream, thus being allowed to reduce the memory usage while sending partial updates to the client. One of the most promising libraries is *Apache DataSketches* [9], which includes the *KLL Sketch* algorithm [79].

Index planet files

An alternative solution could be to create indexes on top of the existing exported OSM PBF files. It could be accomplished by further improving the OSM FDW project or realizing an SQLite Virtual Table to access OSM PBF files and creating ID, version and spatial index when scanning the file for the first time. Each index could then be shared publicly, and other clients could require specific portions of the planet file based on that (as the HTTP server serving the planet file with support for ranged requests, as well as the corresponding torrent). That solution could be a nice solution for lightweight analysis on low powered/low memory machines, but would require an important effort to be developed. Parsing PBF files directly could pave the way to doing analysis on users and tags too.

Faster rendering

Node clustering can make the browser slow when thousands of nodes are shown on the map. The company MapBox [88] developed innovative ways to represent big amount of data on a map quickly, by creating optimized libraries and/or relying on the capabilities of modern web browsers, such as vector tiles and WebGL [179]. Their most famous library is *mapbox-gl-js* [87], which provides clustering and heatmaps generation. While its clustering capability is flexible enough to be used by *Is OSM up-to-date?* (as it supports custom styling), its heatmap generation (which provides a smoother and more pleasant interaction) is not yet there, as there is partial support for using the node values instead of their density when colouring the heatmap [143].

A different solution, compatible with the existing Leaflet map, would be to use a different MapBox library, called *SuperCluster* [168], capable of rendering even millions of nodes.

The Leaflet community is also proposing to bring such type of capabilities to Leaflet itself, and adopting newer JavaScript standards and conventions [97].

Faster computation

One of the possible future improvements that could bring better performances without any major code change would be the adoption of Python 3.11, which greatly reduces computational time under specific scenarios [183]. There is no stable version at the time of writing, and it has not been possible to test the release candidate since the software relies on dependencies that are not yet compatible.

Code profiling has been used during development, to highlight the sections that required

most of the processing time and resources, but it could also hint at possible computations that could be implemented outside Python, using languages such as C, C++ or Rust for example.

Computing and storing indexes along cached data could also provide a performance improvement when filtering the data using custom boundaries.

6.2.4. Hosting

Availability improvements

Even if caching reduces the impact of Ohsome API being down from time to time, it is worth thinking about ways to properly address this issue.

Since Ohsome can be hosted on premises, it would be greatly beneficial if another institution or community would host its replica because it would allow developing a load balancing mechanism and fallback when one of the instances is not available.

Bibliography

[1] 262588213843476. OSM node history to SQLite. Gist. URL: https://gist.github.com/frafra/56650708033b6dd5bb3906827d0e58cc.

[2] 262588213843476. Processing databases generated with osh2sqlite and MMQGIS for Dar es Salaam. Gist. URL: https://gist.github.com/frafra/0e9d94985858706b770e5b551b892d22.

[3] Achavi - Augmented OSM Change Viewer [attic]. URL: https://overpass-api.de/achavi/.

[4] Achavi source code. URL: https://github.com/nrenner/achavi/.

[5] Add /users/count/groupBy/element endpoint · Issue #197 · GIScience/ohsome-api. GitHub. URL: https://github.com/GIScience/ohsome-api/issues/197.

[6] J. Almendros-Jiménez and A. Becerra-Terón. Analyzing the Tagging Quality of the Spanish OpenStreetMap. *ISPRS International Journal of Geo-Information*, 7(8), 2018. DOI: 10.3390/ijgi7080323.

[7] Amazon Athena - Serverless Interactive Query Service - Amazon Web Services. Amazon Web Services, Inc. URL: https://aws.amazon.com/athena/.

[8] V. Antoniou and A. Skopeliti. Measures and indicators of VGI quality: An overview. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W5:345–351, 2015. DOI: 10.5194/isprsannals-II-3-W5-345-2015.

[9] Apache DataSketches Core C++ Library Component, The Apache Software Foundation, June 25, 2022. URL: https://github.com/apache/datasketches-cpp.

[10] API - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/API.

[11] API Usage policy. URL: https://operations.osmfoundation.org/policies/api/.

[12] API v0.6 - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/API_v0.6#History:_GET_.2Fapi.2F0.6.2F.5Bnode.7Cway.7Crelation.5D.2F.23id.2Fhistory.

[13] Application Monitoring and Error Tracking Software. Sentry. URL: https://sentry.io/welcome/.

[14] Async http client/server framework, aio-libs, May 15, 2022. URL: https://github.com/aio-libs/aiohttp.

[15] M. Auer, M. Eckle, S. Fendrich, L. Griesbaum, F. Kowatsch, S. Marx, M. Raifer, M. Schott, R. Troilo, and A. Zipf. Towards Using the Potential of OpenStreetMap History for Disaster Activation Monitoring. In *Proceedings of the 15th ISCRAM Conference*, pages 317–325, 2018.

[16] C. Barrington-Leigh and A. Millard-Ball. The world's user-generated road map is more than 80% complete. *PLOS ONE*, 12(8):e0180698, 2017. DOI: 10.1371/journal.pone.0180698.

[17] C. Barron, P. Neis, and A. Zipf. A Comprehensive Framework for Intrinsic OpenStreetMap Quality Analysis: A Comprehensive Framework for Intrinsic OpenStreetMap Quality Analysis. *Transactions in GIS*, 18(6):877–895, 2014. DOI: 10.1111/tgis.12073.

[18] A. Bona. Adrianulbona/osm-parquetizer, May 3, 2022. URL: https://github.com/adrianulbona/osm-parquetizer.

[19] Brave Mappers of LA County. URL: http://mvexel.github.io/bravemappers/.

[20] M. A. Brovelli, M. Minghini, M. E. Molinari, and G. Zamboni. Positional accuracy assessment of the OpenStreetMap buildings layer through automatic homologous pairs detection: the method and a case study. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. XXIII ISPRS Congress, Commission II (Volume XLI-B2) - 12-19 July 2016, Prague, Czech Republic, volume XLI-B2, pages 615–620. Copernicus GmbH, June 8, 2016. DOI: 10.5194/isprs-archives-XLI-B2-615-2016. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B2/615/2016/.

[21] M. A. Brovelli, M. Minghini, M. Molinari, and P. Mooney. Towards an automated comparison of OpenStreetMap with authoritative road datasets. *Transactions in GIS*, 21(2):191–206, 2017. DOI: 10.1111/tgis.12182.

[22] BuildKit, Moby, July 8, 2022. URL: https://github.com/moby/buildkit.

[23] R. Canavosio-Zuzelski, P. Agouris, and P. Doucette. A photogrammetric approach for assessing positional accuracy of OpenStreetMap\copyright roads.

ISPRS International Journal of Geo-Information, 2(2):276–301, 2013. DOI: 10.3390/ijgi2020276.

[24] Changeset - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Changeset.

[25] Check_date | Keys | OpenStreetMap Taginfo. URL: https://taginfo.openstreetmap.org/keys/check_date.

[26] B. Ciepłuch, P. Mooney, and A. C. Winstanley. Building Generic Quality Indicators for OpenStreetMap. In 19th Annual GIS Research UK (GISRUK), Apr. 2011. URL: http://www.port.ac.uk/special/gisruk2011/.

[27] Commit Activity · frafra/is-osm-uptodate. GitHub. URL: https://github.com/frafra/is-osm-uptodate.

[28] Continuous Integration and Delivery. CircleCI. URL: https://circleci.com/.

[29] P. Corcoran, P. Mooney, and M. Bertolotto. Analysing the growth of OpenStreetMap networks. *Spatial Statistics*, 3:21–32, 2013. DOI: 10.1016/j.spasta.2013.01.002.

[30] Creative Commons — Attribution-ShareAlike 2.0 Generic — CC BY-SA 2.0. URL: https://creativecommons.org/licenses/by-sa/2.0/.

[31] CryptPad, XWiki labs, June 24, 2022. URL: https://github.com/xwiki-labs/cryptpad.

[32] Curl/curl, curl, June 26, 2022. URL: https://github.com/curl/curl.

[33] Dependabot. GitHub. URL: https://github.com/dependabot.

[34] R. Devillers and R. Jeansoulin. Spatial data quality: concepts. In R. Devillers and R. Jeansoulin, editors, *Fundamentals of Spatial Data Quality*, pages 31–42. Wiley Online Library, Hoboken, New Jersey, 2010.

[35] M. Dittus, G. Quattrone, and L. Capra. Mass Participation During Emergency Response: Event-centric Crowdsourcing in Humanitarian Mapping. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, pages 1290–1303, New York, NY, USA. Association for Computing Machinery, Feb. 25, 2017. ISBN: 978-1-4503-4335-0. DOI: 10.1145/2998181.2998216. URL: https://doi.org/10.1145/2998181.2998216.

[36] Docker CLI, Docker, June 25, 2022. URL: https://github.com/docker/cli.

[37] Docker Compose v2, Docker, May 15, 2022. URL: https://github.com/docker/compose.

[38] Docker-OSM, Kartoza Open Source Geospatial Solutions, May 13, 2022. URL: https://github.com/kartoza/docker-osm.

[39] Elements - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Elements.

[40] ESLint, ESLint, May 15, 2022. URL: https://github.com/eslint/eslint.

[41] J. Estima and M. Painho. Investigating the potential of OpenStreetMap for land use/land cover production: A case study for continental Portugal. In *OpenStreetMap in GIScience*, pages 273–293. Springer, Cham, 2015.

[42] H. Fan, A. Zipf, Q. Fu, and P. Neis. Quality assessment for building footprints data on OpenStreetMap. *International Journal of Geographical Information Science*, 28(4):700–719, 2014. DOI: 10.1080/13658816.2013.867495.

[43] Flask, Pallets, May 15, 2022. URL: https://github.com/pallets/flask.

[44] Font Awesome. URL: https://fontawesome.com.

[45] C. C. Fonte, J. A. Patriarca, M. Minghini, V. Antoniou, L. See, and M. A. Brovelli. Using OpenStreetMap to create Land Use and Land Cover maps: Development of an application. In *Volunteered Geographic Information and the Future of Geospatial Data*, pages 113–137. IGI Global, Hershey, Pennsylvania, 2017.

[46] G. Foody, S. Fritz, C. C. Fonte, L. Bastin, A.-M. Olteanu-Raimond, P. Mooney, L. See, V. Antoniou, H.-Y. Liu, M. Minghini, and R. Vatseva. Mapping and the Citizen Sensor. In G. Foody, S. Fritz, P. Mooney, A.-M. Olteanu-Raimond, C. C. Fonte, and V. Antoniou, editors, *Mapping and the Citizen Sensor*, pages 1–12. Ubiquity Press, London, 2017.

[47] C. Fram, K. Chistopoulou, and C. Ellul. Assessing the quality of OpenStreetMap building data and searching for a proxy variable to estimate OSM building data completeness. In *Proceedings of the GIS Research UK (GISRUK)*, pages 195–205, 2015.

[48] F. Frassinelli. [Talk-it] Is OSM up-to-date? Nuova versione e sondaggio, E-mail, Lun 4 Lug 2022 08:10:26 UTC. URL: https://lists.openstreetmap.org/pipermail/talk-it/2022-July/074004.html.

[49] F. Frassinelli. Intrinsic assessment of the temporal accuracy, up-to-dateness, lineage and thematic accuracy of OpenStreetMap. July 28–30, 2018.

[50] F. Frassinelli. Is OSM up-to-date?, Jan. 24, 2022. URL: https://github.com/frafra/is-osm-uptodate.

[51] F. Frassinelli. Un approccio open source per la valutazione intrinseca di accuratezza tematica, accuratezza temporale, aggiornamento e lignaggio di OSM. Feb. 19–22, 2018.

[52] GeoJSON — GDAL documentation. URL: https://gdal.org/drivers/vector/geojson.html.

[53] J.-F. Girres and G. Touya. Quality assessment of the French OpenStreetMap dataset. *Transactions in GIS*, 14(4):435–459, 2010. DOI: 10.1111/j.1467-9671.2010.01203.x.

[54] GNU Affero General Public License - GNU Project - Free Software Foundation. URL: https://www.gnu.org/licenses/agpl-3.0.en.html.

[55] M. F. Goodchild and L. Li. Assuring the quality of volunteered geographic information. *Spatial statistics*, 1:110–120, 2012. DOI: 10.1016/j.spasta.2012.03.002.

[56] A. Graser, M. Straub, and M. Dragaschnig. Is OSM good enough for vehicle routing? A study comparing street networks in Vienna. In *Progress in Location-Based Services 2014*, pages 3–17. Springer, Cham, 2015.

[57] S. Gröchenig, R. Brunauer, and K. Rehrl. Estimating completeness of VGI datasets by analyzing community activity over time periods. In J. Huerta, S. Schade, and C. Granell, editors, *Connecting a Digital Europe through Location and Place*, pages 3–18. Springer, Cham, 2014.

[58] Gunicorn - Python WSGI HTTP Server for UNIX. URL: https://gunicorn.org/.

[59] M. Hacar, B. Kılıç, and K. Şahbaz. Analyzing OpenStreetMap Road Data and Characterizing the Behavior of Contributors in Ankara, Turkey. *ISPRS International Journal of Geo-Information*, 7(10):400, 2018. DOI: 10.3390/ijgi7100400.

[60] M. Haklay. How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. *Environment and planning B: Planning and design*, 37(4):682–703, 2010. DOI: 10.1068/b35097.

[61] R. Hecht, C. Kunze, and S. Hahmann. Measuring completeness of building footprints in OpenStreetMap over space and time. *ISPRS International Journal of Geo-Information*, 2(4):1066–1091, 2013. DOI: 10.3390/ijgi2041066.

[62] L. Hilaiel. Lloyd/yajl, May 12, 2022. URL: https://github.com/lloyd/yajl.

[63] Home - Docker. May 10, 2022. URL: https://www.docker.com/.

[64] How did you contribute to OpenStreetMap ? URL: http://hdyc.neis-one.org/.

[65] HTTPie: human-friendly CLI HTTP client for the API era, HTTPie, June 26, 2022. URL: https://github.com/httpie/httpie.

[66] Hugapi/hug, Hug API Framework, May 11, 2022. URL: https://github.com/hugapi/hug.

[67] Implementing /elementsFullHistory · GIScience/ohsome-api@c3c5c5a. GitHub. URL: https://github.com/GIScience/ohsome-api/commit/c3c5c5a7b32fcac1a50e39f9711c4389d722f788.

[68] Imposm, Omniscale, May 11, 2022. URL: https://github.com/omniscale/imposm3.

[69] Initial commit. · GIScience/ohsome-api@0027363. GitHub. URL: https://github.com/GIScience/ohsome-api/commit/0027363e821efcb4a60e63ee42dc46cb718c501f.

[70] Initial import · frafra/is-osm-uptodate@771ab66. GitHub. URL: https://github.com/frafra/is-osm-uptodate/commit/771ab66c309822c7057eaea1ed67f1647a295c4c.

[71] Is OSM up-to-date? URL: https://is-osm-uptodate.frafra.eu/#19/45.46423/9.19073.

[72] Is-osm-uptodate | Preceden. URL: https://www.preceden.com/git/is-osm-uptodate/831849/805380d4fabd6107.

[73] ISO 19157:2013. ISO. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/25/32575.html.

[74] Issues · yuzhva/react-leaflet-markercluster. GitHub. URL: https://github.com/yuzhva/react-leaflet-markercluster.

[75] S. P. Jackson, W. Mullen, P. Agouris, A. Crooks, A. Croitoru, and A. Stefanidis. Assessing completeness and spatial error of features in volunteered geographic information. *ISPRS International Journal of Geo-Information*, 2(2):507–530, 2013. DOI: 10.3390/ijgi2020507.

[76] Jazzband/pip-tools, Jazzband, July 8, 2022. URL: https://github.com/jazzband/pip-tools.

[77] J. Jokar Arsanjani, M. Helbich, M. Bakillah, and L. Loos. The emergence and evolution of OpenStreetMap: a cellular automata approach. *International Journal of Digital Earth*, 8(1):76–90, 2015. DOI: 10.1080/17538947.2013.847125.

[78] J. Jokar Arsanjani, A. Zipf, P. Mooney, and M. Helbich. An introduction to OpenStreetMap in geographic information science: Experiences, research, and applications. In J. Jokar Arsanjani, A. Zipf, P. Mooney, and M. Helbich, editors, *OpenStreetMap in GIScience*, pages 1–15. Springer, Cham, 2015.

[79] Z. Karnin, K. Lang, and E. Liberty. Optimal Quantile Approximation in Streams. Apr. 5, 2016. DOI: 10.48550/arXiv.1603.05346. arXiv: 1603.05346 [cs]. URL: http://arxiv.org/abs/1603.05346.

[80] C. Keßler and R. T. A. De Groot. Trust as a proxy measure for the quality of volunteered geographic information in the case of OpenStreetMap. In D. Vandenbroucke, B. Bucher, and C. Joep, editors, *Geographic Information Science at the Heart of Europe*, pages 21–37. Springer, Cham, 2013.

[81] Key:check_date - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Key:check_date.

[82] Leaflet.markercluster, Leaflet, June 5, 2022. URL: https://github.com/Leaflet/Leaflet.markercluster.

[83] Leaflet/Leaflet, Leaflet, May 15, 2022. URL: https://github.com/Leaflet/Leaflet.

[84] C. Leifer. Coleifer/walrus, May 14, 2022. URL: https://github.com/coleifer/walrus.

[85] LGTM - Continuous security analysis. URL: https://lgtm.com/.

[86] Map features - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Map_features.

[87] Mapbox/mapbox-gl-js: Interactive, thoroughly customizable maps in the browser, powered by vector tiles and WebGL. URL: https://github.com/mapbox/mapbox-gl-js/.

[88] Maps, geocoding, and navigation APIs & SDKs | Mapbox. URL: https://www.mapbox.com/.

[89] D. Marakasov. Jsonslicer - stream JSON parser, May 7, 2022. URL: https://github.com/AMDmi3/jsonslicer.

[90] Mercantile, Mapbox, May 22, 2022. URL: https://github.com/mapbox/mercantile.

[91] M. Minghini, M. A. Brovelli, and F. Frassinelli. An open source approach for the intrinsic assessment of the temporal accuracy, up-to-dateness and lineage of OpenStreetMap. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. FOSS4G 2018 – Academic Track (Volume XLII-4/W8) - 29-31 August 2018, Dar Es Salaam, Tanzania, volume XLII-4-W8, pages 147–154. Copernicus GmbH, July 11, 2018. DOI: 10.5194/isprs-archives-XLII-4-W8-147-2018. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W8/147/2018/.

[92] M. Minghini and F. Frassinelli. OpenStreetMap history for intrinsic quality assessment: Is OSM up-to-date? *Open geospatial data, softw. stand.*, 4(1):1–17, 1, Dec. 2019. ISSN: 2363-7501. DOI: 10.1186/s40965-019-0067-x. URL: https://opengeospatialdata.springeropen.com/articles/10.1186/s40965-019-0067-x.

[93] M. Minghini, D. Oxoli, F. Frassinelli, and M. A. Brovelli. Intrinsic assessment of OpenStreetMap contribution patterns through Exploratory Spatial Data

Analysis. Sept. 16, 2019. DOI: 10.5281/zenodo.3387683. URL: https://zenodo.org/record/3387683.

[94] M. Minghini, A. Sarretta, F. Lupia, M. Napolitano, A. Palmas, and L. Delucchi. Collaborative mapping response to disasters through OpenStreetMap: the case of the 2016 Italian earthquake. *GEAM - Geoengineering Environment and Mining*, 151(2):21–26, 2017.

[95] F.-B. Mocnik. OSMvis - OpenStreetMap Visualization, Sept. 19, 2021. URL: https://github.com/mocnik-science/osm-vis.

[96] F.-B. Mocnik, A. Mobasheri, and A. Zipf. Open source data mining infrastructure for exploring and analysing OpenStreetMap. *Open Geospatial Data, Software and Standards*, 3(1):7, May 28, 2018. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0047-6. URL: https://doi.org/10.1186/s40965-018-0047-6.

[97] Modern JavaScript · Issue #7615 · Leaflet/Leaflet. URL: https://github.com/Leaflet/Leaflet/issues/7615.

[98] P. Mooney and P. Corcoran. Analysis of Interaction and Co-editing Patterns amongst OpenStreetMap Contributors: Analysis of Interaction and Co-editing Patterns amongst OpenStreetMap Contributors. *Transactions in GIS*, 18(5):633–659, 2014. DOI: 10.1111/tgis.12051.

[99] P. Mooney and M. Minghini. A Review of OpenStreetMap Data. In G. Foody, S. Fritz, P. Mooney, A.-M. Olteanu-Raimond, C. C. Fonte, and V. Antoniou, editors, *Mapping and the Citizen Sensor*, pages 37–59. Ubiquity Press, London, 2017.

[100] B. I. Muttaqien, F. O. Ostermann, and R. L. G. Lemmens. Modeling aggregated expertise of user contributions to assess the credibility of OpenStreetMap features. *Transactions in GIS*, 22(3):823–841, 2018. DOI: 10.1111/tgis.12454.

[101] A. Nasiri, R. Ali Abbaspour, A. Chehreghan, and J. Jokar Arsanjani. Improving the Quality of Citizen Contributed Geodata through Their Historical Contributions: The Case of the Road Network in OpenStreetMap. *ISPRS International Journal of Geo-Information*, 7(7):253, 2018. DOI: 10.3390/ijgi7070253.

[102] P. Neis, D. Zielstra, and A. Zipf. The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007–2011. *Future Internet*, 4(1):1–21, 2011. DOI: 10.3390/fi4010001.

[103] Npm - a JavaScript package manager, npm, June 25, 2022. URL: https://github.com/npm/cli.

[104] Ohsome - Dashboard. URL: https://ohsome.org/apps/dashboard/.

[105] Ohsome | Heidelberg Institute for Geoinformation Technology. URL: https://heigit.org/big-spatial-data-analytics-en/ohsome/.

[106] ohsomeHeX - OSM History Explorer. URL: https://hex.ohsome.org/#/amenity_clinic_healthcare_clinic_ptpl/2022-05-01T00:00:00Z/2/0/0.

[107] Open Data Commons Open Database License (ODbL) v1.0 — Open Data Commons: legal tools for open data. URL: https://opendatacommons.org/licenses/odbl/1-0/.

[108] Open Database License - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Open_Database_License.

[109] OpenStreetMap. OpenStreetMap. URL: https://www.openstreetmap.org/.

[110] OpenStreetMap Blog. 100 million edits to OpenStreetMap. Feb. 25, 2021. URL: https://blog.openstreetmap.org/2021/02/25/100-million-edits-to-openstreetmap/.

[111] OpenStreetMap live edits - live.openstreetmap.fr. URL: http://live.openstreetmap.fr/.

[112] OpenStreetMap on AWS - Registry of Open Data on AWS. URL: https://registry.opendata.aws/osm/.

[113] OpenStreetMap Statistics. URL: https://planet.openstreetmap.org/statistics/data_stats.html.

[114] OpenStreetMap Taginfo. URL: https://taginfo.openstreetmap.org/.

[115] OSM Deep History, OSM Lab, May 5, 2022. URL: https://github.com/osmlab/osm-deep-history.

[116] OSM History Viewer - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=OSM_History_Viewer.

[117] OSM Latest Changes, OSM Lab, Apr. 3, 2022. URL: https://github.com/osmlab/latest-changes.

[118] Osm tag history. URL: https://taghistory.raifer.tech/.

[119] OSM XML - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=OSM_XML.

[120] Osm-analytics: data analysis tool frontend, Humanitarian OpenStreetMap Team, May 1, 2022. URL: https://github.com/hotosm/osm-analytics.

[121] Osm-data.skobbler.net. Sept. 9, 2019. URL: https://web.archive.org/web/20190909091258/http://osm-data.skobbler.net/.

[122] Osm2pgsql, OpenStreetMap on GitHub, May 13, 2022. URL: https://github.com/openstreetmap/osm2pgsql.

[123] Osm2pgsql/benchmarks - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Osm2pgsql/benchmarks.

[124] OSMCha. OSMCha. URL: https://osmcha.org/.

[125] OSMesa, Azavea, Feb. 11, 2022. URL: https://github.com/azavea/osmesa.

[126] OSMF Licence Working Group - White Paper on the Introduction of the General Data Protection Regulation. URL: https://wiki.openstreetmap.org/w/images/8/88/GDPR_Position_Paper.pdf.

[127] Osmium Command Line Tool, osmcode, May 15, 2022. URL: https://github.com/osmcode/osmium-tool.

[128] Osmlab/show-me-the-way: See OSM edits happen in real time. URL: https://github.com/osmlab/show-me-the-way.

[129] Osmrmhv/osmrmhv, osmrmhv, May 6, 2022. URL: https://github.com/osmrmhv/osmrmhv.

[130] OSMstats - Statistics of the free wiki world map. URL: https://osmstats.neis-one.org/.

[131] Overpass API - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Overpass_API.

[132] Overpass API/Language Guide - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Overpass_API/Language_Guide.

[133] PDM - Python Development Master, Python Development Master(PDM), June 25, 2022. URL: https://github.com/pdm-project/pdm.

[134] PEP 517 – A build-system independent format for source trees | peps.python.org. URL: https://peps.python.org/pep-0517/.

[135] PEP 582 – Python local packages directory | peps.python.org. URL: https://peps.python.org/pep-0582/.

[136] PEP 621 – Storing project metadata in pyproject.toml | peps.python.org. URL: https://peps.python.org/pep-0621/.

[137] V. Pikulik. OSM PBF Foreign Data Wrapper, Apr. 16, 2022. URL: https://github.com/vpikulik/postgres_osm_pbf_fdw.

[138] Planet.osm - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Planet.osm.

[139] Planet.osm/full - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Planet.osm/full.

[140] Poetry: Dependency Management for Python, Poetry, July 8, 2022. URL: https://github.com/python-poetry/poetry.

[141] Pre-commit/pre-commit, pre-commit, May 15, 2022. URL: https://github.com/pre-commit/pre-commit.

[142] Project-EPIC/epic-osm. URL: https://github.com/Project-EPIC/epic-osm.

[143] Proposal: Heatmap layer type · Issue #4756 · mapbox/mapbox-gl-js. GitHub. URL: https://github.com/mapbox/mapbox-gl-js/issues/4756.

[144] Psf/black, Python Software Foundation, May 15, 2022. URL: https://github.com/psf/black.

[145] PyCQA/flake8, Python Code Quality Authority, May 15, 2022. URL: https://github.com/PyCQA/flake8.

[146] PyCQA/isort, Python Code Quality Authority, May 15, 2022. URL: https://github.com/PyCQA/isort.

[147] PyGEOS, pygeos, June 24, 2022. URL: https://github.com/pygeos/pygeos.

[148] Pytest-dev/pytest, pytest-dev, June 26, 2022. URL: https://github.com/pytest-dev/pytest.

[149] Qgis/QGIS, QGIS, June 5, 2022. URL: https://github.com/qgis/QGIS.

[150] QXOSM. URL: http://xosm.ual.es:8080/qxosm/#!QXOSM.

[151] M. Raifer, R. Troilo, F. Kowatsch, M. Auer, L. Loos, S. Marx, K. Przybill, S. Fendrich, F.-B. Mocnik, and A. Zipf. OSHDB: a framework for spatio-temporal analysis of OpenStreetMap history data. *Open Geospatial Data, Software and Standards*, 4(1):3, Apr. 8, 2019. ISSN: 2363-7501. DOI: 10.1186/s40965-019-0061-3. URL: https://doi.org/10.1186/s40965-019-0061-3.

[152] M. Raifer, R. Troilo, F.-B. Mocnik, and M. Schott. OSHDB - OpenStreetMap History Data Analysis, version 0.7.2, July 2021. DOI: 10.5281/zenodo.4146990. URL: https://github.com/GIScience/oshdb.

[153] rbrundritt. Bing Maps Tile System - Bing Maps. URL: https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system.

[154] React ·, Meta, May 15, 2022. URL: https://github.com/facebook/react.

[155] Redis internals, Redis, May 15, 2022. URL: https://github.com/redis/redis.

[156] Release Version 1.0.0 · GIScience/ohsome-api. GitHub. URL: https://github.com/GIScience/ohsome-api/releases/tag/1.0.0.

[157] Releases · protocolbuffers/protobuf. GitHub. URL: https://github.com/protocolbuffers/protobuf/releases.

[158] O. Roick, J. Hagenauer, and A. Zipf. OSMatrix–grid-based analysis and visualization of OpenStreetMap. In *Proceedings of State of the Map Europe 2011*, pages 44–54, 2011.

[159] S. Sehra, J. Singh, and H. Rai. Using Latent Semantic Analysis to Identify Research Trends in OpenStreetMap. *ISPRS International Journal of Geo-Information*, 6(7):195, 2017. DOI: 10.3390/ijgi6070195.

[160] Selenium. Selenium. URL: https://www.selenium.dev/.

[161] Seleniumbase/SeleniumBase, SeleniumBase, June 26, 2022. URL: https://github.com/seleniumbase/SeleniumBase.

[162] Shapely, shapely, June 25, 2022. URL: https://github.com/shapely/shapely.

[163] Should I use POIs or areas to identify shops? - OSM Help. URL: https://help.openstreetmap.org/questions/22962/should-i-use-pois-or-areas-to-identify-shops.

[164] Sliced Time and Space. URL: https://dev.overpass-api.de/blog/sliced_time_and_space.html.

[165] SQLite Release 3.9.0 On 2015-10-14. URL: https://sqlite.org/releaselog/3_9_0.html.

[166] Stats - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/w/index.php?title=Stats.

[167] Status.ohsome.org. URL: https://status.ohsome.org/.

[168] Supercluster, Mapbox, May 11, 2022. URL: https://github.com/mapbox/supercluster.

[169] Survey | Sondaggio · frafra/is-osm-uptodate Wiki. GitHub. URL: https://github.com/frafra/is-osm-uptodate.

[170] Swagger UI. URL: https://api.ohsome.org/v1/swagger-ui.html.

[171] Tag:highway=steps - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Tag:highway%3Dsteps#Tags_to_use_in_combination.

[172] Y. Tian, Q. Zhou, and X. Fu. An Analysis of the Evolution, Completeness and Spatial Patterns of OpenStreetMap Building Data in China. *ISPRS International Journal of Geo-Information*, 8(1):35, 2019. DOI: 10.3390/ijgi8010035.

[173] Tiled web map. In *Wikipedia*. Mar. 1, 2022. URL: https://en.wikipedia.org/w/index.php?title=Tiled_web_map&oldid=1074646619.

[174] Twbs/bootstrap, Bootstrap, May 15, 2022. URL: https://github.com/twbs/bootstrap.

[175] Unbit. Unbit/uwsgi, May 14, 2022. URL: https://github.com/unbit/uwsgi.

[176] Y. Uzhva. React leaflet markercluster, May 12, 2022. URL: https://github.com/yuzhva/react-leaflet-markercluster.

[177] H. Veregin. Data quality parameters. In *Geographical Information Systems*, pages 177–189. John Wiley & Sons, Hoboken, New Jersey, 1999.

[178] Visualize-change, Humanitarian OpenStreetMap Team, May 15, 2022. URL: https://github.com/hotosm/visualize-change.

[179] WebGL. The Khronos Group. July 19, 2011. URL: https://www.khronos.org//.

[180] Webpack, webpack, May 15, 2022. URL: https://github.com/webpack/webpack.

[181] Welcome to the documentation of the ohsome API! — ohsome API 1.6.3 documentation. URL: https://docs.ohsome.org/ohsome-api/stable/.

[182] Wget - GNU Project - Free Software Foundation. URL: https://www.gnu.org/software/wget/.

[183] What's New In Python 3.11 — Python 3.11.0b3 documentation. URL: https://docs.python.org/3.11/whatsnew/3.11.html#faster-cpython.

[184] WHODIDIT: OpenStreetMap Changeset Analyzer. URL: https://simon04.dev.openstreetmap.org/whodidit/.

[185] P. Zhao, T. Jia, K. Qin, J. Shan, and C. Jiao. Statistical analysis on the evolution of OpenStreetMap road networks in Beijing. *Physica A: Statistical Mechanics and its Applications*, 420:59–72, 2015. DOI: 10.1016/j.physa.2014.10.076.

[186] Zoom levels - OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Zoom_levels.

List of Figures

3.1 FOSS4G-IT/Merge-IT 2018 – North Italy, contributor density 22
3.2 FOSS4G 2018 – Dar es Salaam, average last edit 23
3.3 SOTM 2018 – Milan, average creation time 24

4.1 Web app – Zoom level 19, individual nodes with pop-up 30
4.2 Web app – Zoom level 17, clustered nodes 31
4.3 Web app – Zoom level 16, tiles 32
4.4 Web app – Shops within Milano administrative boundary 33
4.5 Web app – Shops within Milano administrative boundary, scrolled down . . 34
4.6 QGIS showing *Is OSM up-to-date?* tile layer 37
4.7 *Piazza del Duomo, Milano* shown by Nominatim 42
4.8 *Municipio 1, Milano* shown by Nominatim 44
4.9 *Milano*, shown by Nominatim 44
4.10 *Città Metropolitana di Milano*, shown by Nominatim 46

List of Graphs

4.1 Development timeline 38
4.2 Line added and removed, by week 39
4.3 Software performances for *Piazza del Duomo, Milano* 43
4.4 Software performances for *Municipio 1, Milano* 43
4.5 Software performances for *Milano* city 45
4.6 Software performances for *Città Metropolitana di Milano* 46
4.7 Software architecture, conceptual map 47

5.1 Scores users gave to each feature 53
5.2 Survey – Have you tried it before? 55
5.3 Survey – Do you think it has been improved recently? 56
5.4 Survey – Is it useful? 56
5.5 Survey – Would you use it in the future? 56

List of Tables

4.1 Lines of code, by language and type 38

Acknowledgements

Special thanks to professor Maria Brovelli and PhD Marco Minghini, as their contagious passion for geoinformatics and OpenStreetMap have been crucial in my choice to study geoinformatics engineering, which changed my life for the better.

Thanks should also go to my mother Daniela and my father Roberto, which supported me during all these years, and to my partner, Angelica.