Information and Software Technology 138 (2021) 106631

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Metamorphic testing of OpenStreetMap ☆

Jesús M. Almendros-Jiménez [a], Antonio Becerra-Terón [a], Mercedes G. Merayo [b], Manuel Núñez [b,*]

[a] Department of Informatics, Carretera de Sacramento s/n, University of Almería, 04120, Almería, Spain
[b] Institute of Knowledge Technology, Universidad Complutense de Madrid, 28040, Madrid, Spain

ARTICLE INFO

Keywords:
Metamorphic testing
Quality of maps
OpenStreetMap

ABSTRACT

Context: OpenStreetMap represents a collaborative effort of many different and unrelated users to create a free map of the world. Although contributors follow some general guidelines, unsupervised additions are prone to include erroneous information. Unfortunately, it is impossible to automatically detect most of these issues because there does not exist an *oracle* to evaluate whether the information is correct or not. Metamorphic testing has shown to be very useful in assessing the correctness of very heterogeneous artifacts when oracles are not available.

Objective: The main goal of our work is to provide a (fully implemented) framework, based on metamorphic testing, that will support the analysis of the information provided in OpenStreetMap with the goal of detecting *faulty* information.

Method: We defined a general metamorphic testing framework to deal with OpenStreetMap. We identified a set of *good* metamorphic relations. In order to have as much automation as possible, we paid special attention to the automatic selection of *follow-up inputs* because they are fundamental to diminish manual testing. In order to assess the usefulness of our framework, we applied it to analyze maps of four cities in different continents. The rationale is that we would be dealing with different problems created by different contributors.

Results: We obtained experimental evidence that shows the potential value of our framework. The application of our framework to the analysis of the chosen cities revealed errors in all of them and in all the considered categories.

Conclusion: The experiments showed the usefulness of our framework to identify potential issues in the information appearing in OpenStreetMap. Although our metamorphic relations are very helpful, future users of the framework might identify other relations to deal with specific situations not covered by our relations. Since we provide a general pattern to define metamorphic relations, it is relatively easy to extend the existing framework. In particular, since all our metamorphic relations are implemented and the code is freely available, users have a pattern to implement new relations.

1. Introduction

Software Testing [1,2] is the most widely used validation technique to increase the confidence on the correctness of complex systems. Essentially, testing consists in the application of a set of inputs to the System Under Test (SUT), the observation of the produced outputs and the decision on whether the observed outputs are the expected ones. This last step is particularly interesting. Traditionally, the tester was in charge of (mainly manually) deciding whether the outputs showed a failure. In order to automatize this task, it is necessary to have an *oracle*: a (formal) procedure to decide that the observed outputs are indeed the expected ones. However, there are many situations where testers suffer the *oracle problem* [3]: either oracles are not available [4] or it is very costly, in computational terms, to call the (available) oracle. A good solution to analyze an SUT without knowing whether an isolated result is correct consists in jointly studying the relation between several inputs and the corresponding outputs. Essentially, this is the idea behind *Metamorphic Testing* [5–7] (MT), which has been shown to be a very useful tool to ameliorate the oracle problem [8].

As we have already hinted, the main idea behind the definition of an MT framework is to establish *Metamorphic Relations* (MRs) between

☆ This work has been supported by the State Research Agency (AEI) of the Spanish Ministry of Science and Innovation under grants PID2019-104735RB-C42 (SAFER) and RTI2018-093608-B-C31 (FAME); the Region of Madrid, Spain under grant S2018/TCS-4314 (FORTE-CM) co-funded by EIE Funds of the European Union; the Region of Madrid — Complutense University of Madrid, Spain (grant PR65/19-22452).

* Corresponding author.
E-mail addresses: jalmen@ual.es (J.M. Almendros-Jiménez), abecerra@ual.es (A. Becerra-Terón), mgmerayo@fdi.ucm.es (M.G. Merayo), mn@sip.ucm.es (M. Núñez).

https://doi.org/10.1016/j.infsof.2021.106631
Received 27 November 2020; Received in revised form 15 March 2021; Accepted 12 May 2021
Available online 17 May 2021

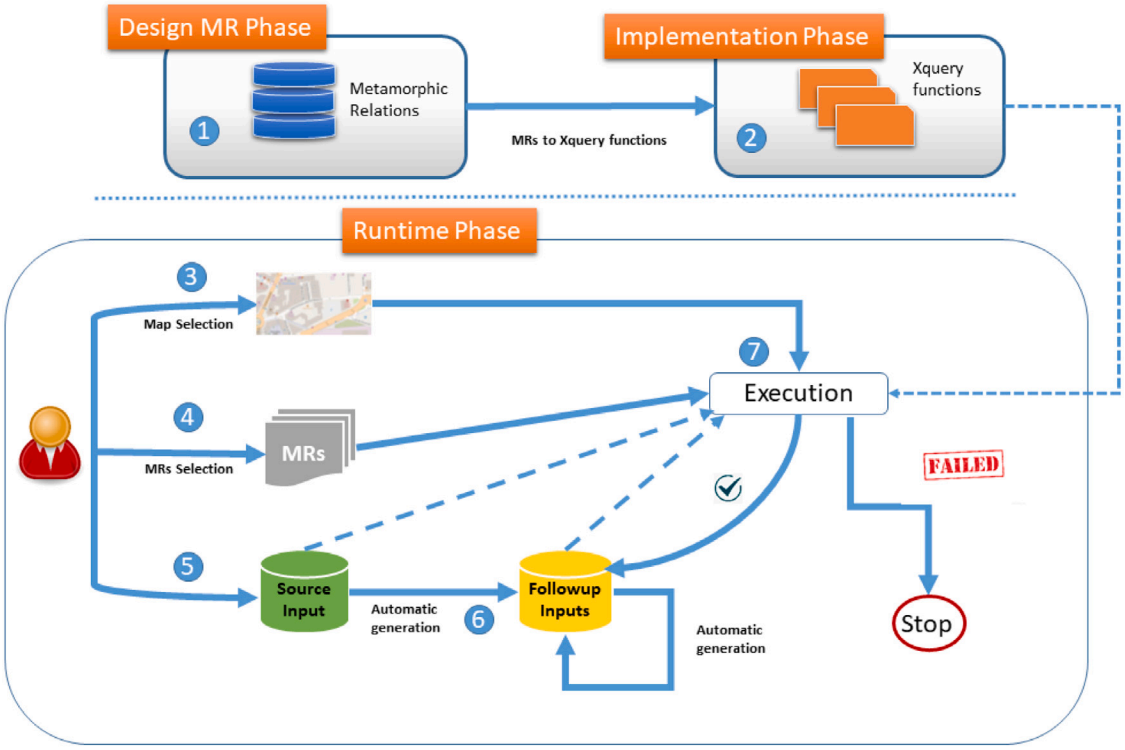


Fig. 1. Architecture of our MT framework.

inputs and outputs. Intuitively, we apply inputs to the SUT, observe the produced outputs and check whether any of the available MRs has been violated. The classical example to motivate the usefulness of MT is the implementation of a sine function. Even though we do not have an oracle, we can *indirectly* detect that a certain implementation P of this function is faulty. For example, we might not know the value that P should return for a certain real value x, but if P returns different values for x and $x + 2 \cdot \pi$, then we know that the implementation is faulty. This property is reflected in an MR as follows:

$$\begin{array}{c} MR_1(i_1, i_2, P(i_1), P(i_2)) \\ \Updownarrow \\ (\exists k \in \mathbb{Z} : i_2 = i_1 + k \cdot 2 \cdot \pi) \Rightarrow P(i_1) = P(i_2) \end{array} \quad (1)$$

where $P(x)$ denotes the application of the input x to the program P.

The second main concept involved in the definition of an MT framework is the distinction between *source* and *follow-up* inputs. Informally, a source input is a value that the tester, after analyzing the expected characteristics of the SUT, considers to be relevant. A follow-up input is an input that will be used, in conjunction with the source input and the corresponding outputs, to conform the application of an MR. For example, a tester working with an implementation of the sine function may consider that 0 is a good value to be exercised. This will be a source input. Using the previous MR, $2 \cdot \pi$ can be (automatically) obtained to be used as a follow-up input. The tester will check whether $MR_1(0, 2 \cdot \pi, P(0), P(2 \cdot \pi))$ holds. Note, and this is an interesting feature that we will extensively exploit in our work, that sometimes a single source input can be used to generate many follow-up inputs. For example, from 0, the tester automatically gets $2 \cdot \pi, 4 \cdot \pi, -2 \cdot \pi, \ldots$, and can check, among others, whether $MR_1(2 \cdot \pi, -2 \cdot \pi, P(2 \cdot \pi), P(-2 \cdot \pi))$ holds.

In this paper we apply MT to find errors in maps. Specifically, we will consider OpenStreetMap (OSM) because it is open source and it is used by many big companies around the world.[1] In Section 2 we review the main characteristics of OSM. In Fig. 1 we graphically present the main components of our approach and next we briefly describe them.

- **Design of MRs**. We have to design a collection of MRs with the intention of detecting potential problems in OSM. The rules take into account how geometric objects should be related to each other. The goal is to establish relations among objects, such as roundabouts, highways or buildings, and reveal geometric mistakes in OSM without requiring external resources, that is, without using an oracle. Our MRs will discover cartographic mistakes, like overlapping and connectivity mistakes, which are crucial for routing purposes.
- **Implementation of MRs**. In this phase, MRs are translated into the XML Database Query Language XQuery. This allows us to apply them to any OSM map of our interest, so that we can detect elements that do not satisfy the established relations.
- **Runtime Execution**. This phase corresponds to the application of the implemented MRs to specific OSM maps. We have four consecutive steps in this phase. First, the tester must provide the OSM map to be analyzed. Then, a subset of the available MRs is selected to be applied to the map. For each of the selected MRs, the tester must provide source inputs corresponding to the elements of the map that will be used as the starting point of the analysis process. Note that for some MRs, these initial inputs can be automatically produced. Then, follow-up inputs are automatically generated from source inputs and the XQuery functions associated to the selected MRs are executed. If an MR is not satisfied, then the process stops and an error is reported. If no error is reported, then former follow-up inputs will play the role of source inputs, new follow-up inputs will be automatically produced, and the process will be iterated until either an error is found or no follow-up inputs are produced (note that repeated follow-up inputs are discarded).

We provide a catalog of MRs that have been shown to be very useful to reveal errors in maps. In particular, they can analyze several orthogonal aspects of OSM. If they are enough to assess the aspects of interest of a potential tester, then the tester does not need to define/implement anything. If testers need to extend the provided framework

[1] https://wiki.openstreetmap.org/wiki/They_are_using_OpenStreetMap.

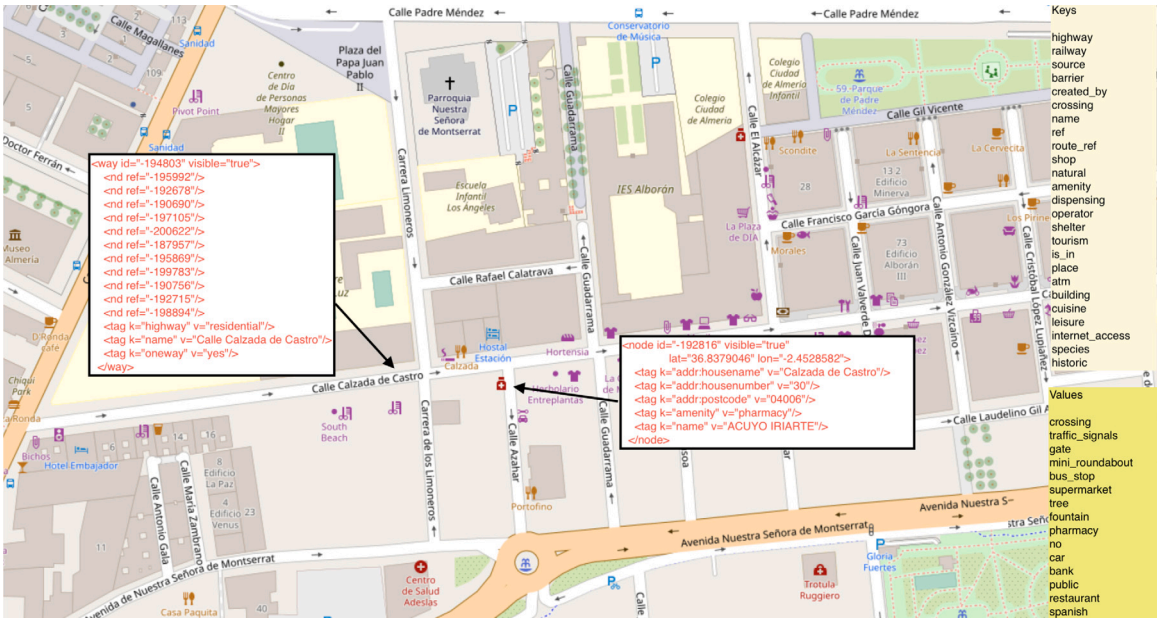


Fig. 2. An example of OSM.

with new MRs, then they can follow the pattern given by the available MRs. Note that the implementation of MRs is straightforward because we use a declarative programming language whose syntax is close to the mathematical definition of our MRs. Finally, the execution phase is the most automated one: the tester must provide source inputs but the rest of the operative is restricted to some trivial interactions with the framework.

We have assessed the usefulness of our framework by applying it to the analysis of OSM maps corresponding to four different cities. Our MT framework was able to find errors in all of them. Moreover, in order to decide whether these errors had been previously noted, we considered the currently most used OSM error checker: *Osmose (Open Street Map Oversight Search Engine)*.[2] As reported later in the paper, we found errors that had not been previously reported by *Osmose*. Moreover, we also considered an official tool provided by OSM, called *notes*, wherein users can report mistakes. Unfortunately, few errors are registered by users through this tool in comparison with the *Osmose* activity. In fact, only one of the errors that we found in our experiments had been reported in *notes*.

The rest of the paper is structured as follows. In Section 2 we informally introduce the main characteristics and concepts of OSM that we will use in our framework. In Section 3 we review related work. In Section 4 we present some concepts that will be used along the paper. Specifically, we will formally define all the elements taking part in OSM and the main notions in metamorphic testing. In Section 5 we define our metamorphic relations. In particular, we describe how follow-up inputs can be generated. We also present details concerning the implementation of these relations. In Section 6 we report on our experiments whose main goal is to assess the usefulness of our framework. We also present and provide answers to our research questions and analyze threats to the validity of our results. Finally, in Section 7 we present our conclusions and discuss some lines for future work.

2. A brief introduction to OSM

OSM [9–11] is a collaborative initiative to create an editable map of the world for sharing free geo-spatial data. OSM is a crowdsourced map in which volunteers contribute either with spatial data (i.e., geometries) or with textual data in the form of *tags*.

Two kinds of geometries are considered in OSM. First, *nodes* are used to represent elements like trees and traffic signals. Second, *ways*, which can be polylines or polygons (closed ways), represent highways and buildings, among others. For example, a set of connected ways with the same name defines a street while a sole closed way represents a building. Tags are (*key, value*) pairs showing that a value is given to a key that represents the name of a certain feature. They serve to provide information about the elements of the map, in particular about points of interest such as museums, hotels, touristic places, etc. In addition, tags also provide useful information about highways, which are also central in OSM to describe routes from one point to another. Fig. 2 shows an excerpt of the map of Almería, Spain. Typically, nodes (for instance, the item on the right-hand side) are described by the *node* label, including the node identifier and coordinates as attributes, together with the sub-label *tag* assigning a set of (key, value) pairs to nodes. Such pairs are frequently used to provide the *name* of the node, as well as pairs (category, subcategory), for instance (*amenity, pharmacy*), being usual to provide other values like postal code and house number. Ways (for instance, the item on the left-hand side) are used to represent geometries in which, additionally to (key, value) pairs, references to the nodes of the geometry are stated. Such nodes are separately declared. Additionally, relations (labeled with the *relation* label) can be stated between sets of nodes/ways to group them and describe more complex structures like bus stops, houses on the same estate, etc.

The OSM community establishes some basic rules that should be followed by contributors.[3] Otherwise, OSM tools and users would find it more difficult to interpret the elements of the map. For instance, in order to indicate a connection between two streets, the last node of one of them must be the first node of the other one. We have a similar situation with a roundabout and the streets it connects (one of the nodes of the roundabout should be a node of the connected streets) or when two streets cross (the point of cross must be a node of both streets). Intersection and overlapping of geometries is, in general, restricted to a few cases. For instance, buildings should not overlap, even if they share one of the sides. Finally, even though nodes can represent a hotel or a museum, the OSM community rules recommend to provide the complete geometry (i.e., a closed way).

[2] http://osmose.openstreetmap.fr/es/map/.

[3] https://wiki.openstreetmap.org/wiki/Editing_Standards_and_Conventions.

3. Related work

Metamorphic Testing (MT) has been applied in different, and very heterogeneous, fields such as cloud systems [12], compilers [13], cryptography [14,15], search engines [16] and simulation [17,18], to name a few. In particular, recent work has applied MT to data validation [19]. This is a line of research related to our work because our main interest is to validate the quality of the data included in a map. Also, it is worth to mention the notion of *metamorphic exploration* [20]. The idea is that a Metamorphic Relation (MR) may sometimes detect situations that do not represent an error but that may help to better understand a certain behavior of the system under test. Our MRs may also identify situations where the detected issue does not represent a *real* geometry error but a labeling error.

The field is mature and there are several complete surveys outlining the main contributions, lines of work and challenges [21,22]. Concerning the scope of this paper, there is not much work on MT of maps. Actually, to the best of our knowledge, there is only one approach to use MT to validate maps [23,24], but the authors focus on relations concerning distances and paths while, as we have already explained, we focus on the information, concerning geometries, included in the map. A recent proposal applies MT to geographic information systems [25]. This approach focuses on the definition of MRs for programs dealing with superficial area calculation. These programs are usually used to calculate the area or volume of an irregular shape. Their MRs are defined on the basis of different artifacts (e.g. requirements, program properties, algorithms) and the effectiveness of the method for detecting failures is evaluated by means of a case study based on mutation testing. Again, there is no relation between this proposal and our framework.

The assessment of the *quality* of OSM maps is crucial to consider OSM as a *reliable* source of geographic data. Whereas the effort of the millions of users has a great value, and the same contributors collaborate in the detection of inappropriate uses of tags and inaccuracies in geometries, a number of *methods* [26–29] and *tools*[4] for quality assurance have been developed during the last years. They report bugs and errors in OSM maps and assist, monitor and visualize OSM contributions.

Data validation systems offer a large number of geometric consistency checks (see Table 1). The already mentioned, and widely used, *Osmose*[5] freely provides the Python code of its validator.[6] Another tool, *Keep Right*,[7] performs several consistency checks, including some geometric consistency checks, and shows them in a map. It provides mechanisms for reporting false positives and for labeling a bug as fixed. *JOSM Validator*, integrated with the JOSM editor,[8] checks data loaded into the editor, highlights errors and warnings, and some automatic fixes are done by request. It checks all objects modified in a session by the user, reporting errors even though the user is not responsible of them. It offers a large number of geometric consistency checks. The Java code of JOSM tests is freely available[9]. *OSM inspector*[10] is an error debugging tool which takes part of the *GeoFabrik* tools. It shows a map with errors classified by different layers: geometry, routing, tagging, places, highways, areas, coastline, addresses, water, public transport stops and public transport routes. It offers a smaller number of geometry consistence checks than the previous mentioned tools. C code of the OSM Inspector is freely available.[11]

Table 1
Geometry errors checked by existing tools.

Tool	Geometry checks
KEEP RIGHT	Almost junctions Bad directions in roundabouts Dead-ended one-ways Intersections without junctions Loops in ways Multiple nodes on the same spot Non-closed areas Overlapping ways Unconnected highways
OSMOSE	Almost junctions Bad directions in roundabouts Boundary intersections Broken highway continuity Duplicate nodes in ways Invalid polygons Non-closed areas Objects overlapping Objects intersection Orphan nodes Overlapping buildings Single nodes in ways Split buildings Unconnected highways/cycleways
JOSM Validator	Crossing ways Duplicated nodes in ways Non-closed areas Overlapping ways Self intersecting ways Unconnected ways Ways connected to areas
OSM Inspector	Duplicate nodes in areas Duplicate segments in areas Self intersecting ways Single nodes in ways Unconnected roads

There are methods achieving an *extrinsic quality analysis* by using an *authoritative dataset*. Such authoritative dataset serves as a *ground truth dataset* to which compare OSM data (that is, they are explicitly using a kind of oracle). In addition, *intrinsic quality analysis* is applied to validate OSM data by assuming some rules of quality [29–38]. Typically [34], these methods use quantitative factors such as the *number of contributors* [30], *number of heavily edited objects* [31] and combinations of such factors, *number of versions* and *number of users*, as well as *confirmations*, *tag corrections* and *rollbacks* [32] to assess the quality of OSM. In some cases, the *history of OSM contributors' updates* is used for the analysis. In this line of work, the *iOSMAnalyzer* tool[12] implements several quality measures and generates a PDF document containing statistics, maps and diagrams which can be used to assess the quality of a selected OSM area [33].

Our MT approach implements an intrinsic quality analysis in which MRs are used to assess the quality of OSM maps. Actually, our proposal opens a new line of research in intrinsic quality evaluation methods based on the following considerations:

1. Most geometry properties that can be assessed by using quality evaluation methods can also be mathematically expressed.
2. Most of them are properties about two or more OSM elements, that is, they are relations among OSM elements. Thus, they can be expressed as MRs.
3. Such relations can be automatically checked.
4. Metamorphic testing provides a suitable framework for the design of (semi-)automatic solutions to check whether such relations hold.

[4] https://wiki.openstreetmap.org/wiki/Quality_assurance.
[5] http://osmose.openstreetmap.fr/es/map/.
[6] https://github.com/osm-fr/osmose-backend/tree/master/plugins.
[7] https://keepright.at/.
[8] https://josm.openstreetmap.de/.
[9] https://josm.openstreetmap.de/browser/josm/trunk/src/org/openstreetmap/josm/data/validation/tests.
[10] https://tools.geofabrik.de/osmi/.
[11] https://github.com/geofabrik/osmi_simple_views.
[12] https://github.com/zehpunktbarron/iOSMAnalyzer.

Let us remark that we have already worked on the topic of OSM [39, 40] and quality assessment of OSM [41]. Specifically, we proposed an extrinsic quality method for evaluating the tagging of Spanish OSM maps based on the comparison with the Taginfo[13] dataset. Our new proposal represents a different approach, intrinsic versus extrinsic, using a completely different underlying technique (because we strongly rely on MT), and supported by a fully implemented and highly automatic framework.

4. Preliminaries

This section introduces the main concepts used in the proposed framework. First, we review the *data primitives* of the topological data structure used by the OSM system to represent the spatial and textual elements of maps. Second, we give formal definitions of the main concepts involved in MT: metamorphic relations and source and follow-up inputs.

Definition 1. The OSM data types are classified in the following categories:

- A *tag* is a pair (k, v) where k and v correspond to a key and a value, respectively. Both of them are given by strings. They must be attached to a node, a way or a relation.[14]
- A *node* is a tuple $n = (id, (lat, long), Tg)$ where $id \in String$ unequivocally identifies the element, lat and $long$ correspond, respectively, to the latitude and longitude of a geographic position according to the World Geodetic System (WGS84) and Tg is the set of tags associated with the node.
- A *way* is a tuple $w = (id, \langle n_1, \ldots, n_s \rangle, Tg)$ where, as in the case of a node, id is the unique identifier of the element, Tg is the set of tags assigned to the way and $\langle n_1, \ldots, n_s \rangle$ is the sequence of nodes that constitute the way. A way represents a *polyline*. If $n_1 = n_s$, then it represents a *polygon*.

Given a set of nodes $\mathcal{N}$, we denote by $\mathcal{W}_{\mathcal{N}}$ the *set of ways* for $\mathcal{N}$, that is, the ways such that they only contain nodes belonging to $\mathcal{N}$.

Given an element (node or way) e, $id(e)$ returns the identifier of the element while $tags(e)$, $keys(e)$ and $vals(e)$ return the set of tags, the set of keys and the set of values associated to it, respectively. Given a node n, the functions $lat(n)$ and $long(n)$ return the latitude and longitude of the node. Given a way w, we denote by $path(w)$ and $set(w)$ its associated sequence and set of nodes, respectively. The functions $first(w)$ and $last(w)$ return the first and last nodes of the way, respectively. Given a node $n \in set(w)$, the function $prev(w, n)$ returns the previous node in $path(w)$. Given a pair of nodes n, n', we denote by $line(n, n')$ the line that pass through the points defined by $(lat(n), long(n))$ and $(lat(n'), long(n'))$ and by $seg(n, n')$ the segment bounded by those points. Finally, given a pair of segments seq_1 and seg_2, we define the function $overlap(seg_1, seg_2)$ as $|seg_1 \cap seg_2| > 1$.

Definition 2. An OSM map is a pair $m = (\mathcal{N}, \mathcal{W}_{\mathcal{N}})$ where $\mathcal{N}$ is the set of nodes of the map and $\mathcal{W}_{\mathcal{N}}$ is the set of ways for $\mathcal{N}$.

A usual restriction on OSM maps is that keys appear at most once in an element. Formally, for all element $e \in \mathcal{N} \cup \mathcal{W}_{\mathcal{N}}$ and key $k \in keys(e)$, we have that there exists a unique value v such that $(k, v) \in tags(e)$.

Abusing the notation, we will write $e \in m$ if $e \in \mathcal{N} \cup \mathcal{W}_{\mathcal{N}}$. During the rest of the paper, if a specific map is not mentioned, then we will assume that we are studying a fix map that we will denote by $\mathcal{M}$.

[13] https://taginfo.openstreetmap.org/.

[14] Although OSM allows the definition of relations, which make possible to group nodes and ways, we restrict our framework to the case of nodes and ways.

In order to test any system, we need to know the kind of *inputs* that we can apply to the system and the kind of *outputs* that we should receive from the system. In our framework, an input describes the nodes and ways, with a certain characteristic, that we would like to find in a given map and the output is the corresponding set.

Definition 3. The following BNF formally describes the syntax of our inputs:

$$
\begin{aligned}
&I := (T, C)\\
&T := *\mid node \mid way\\
&C := \{L\}\\
&L := (V, V) \mid (V, V), L\\
&V := string \mid !string
\end{aligned}
$$

We denote by $\mathcal{I}$ the minimum set including all the expressions that can be derived from the non-terminal symbol I of the previous BNF.

Given an input $i = (T, C)$, we have that $\mathcal{M}(i)$ returns the set of elements in the map $\mathcal{M}$ of type T such that they fulfill the condition given by C.

Although we use a relatively complex syntax to describe inputs, they will usually be very simple and be composed of a couple of clauses. An input is a pair with two components. The first component of the pair indicates whether we are looking for nodes, ways or both (indicated by $*$). The second component is a list of conditions on the strings that keys and values labeling elements must have. The idea is that we are looking for elements having (possibly different) tags matching each of the conditions appearing in the list. Each element of the list will be a pair or strings. The first component of the pair will be associated with keys of the tags and the second one with values. An ! symbol denotes that the string is different from the one appearing in the pair. We allow the wildcard character $*$. This special character matches any string of zero or more characters. If we have a list with a single element, then we simply write (k, v) instead of $\{(k, v)\}$. Finally, for the sake of simplicity, we do not distinguish between capital and lowercase letters in strings.

Example 1. A condition such as $(name, hotel\ *)$ will be matched by those elements having a tag $(name, v)$ such that v matches the pattern $hotel\ *$ (e.g. $Hotel\ California$).

A condition such as $\{(name, !A7), (highway, *)\}$ will be matched by elements being part of a highway whose name is not $A7$. Technically, we will select those elements e such that there exist v_1 and v_2, with $v_1 \neq A7$, such that $(name, v_1), (highway, v_2) \in tags(e)$.

We conclude this section with a formal definition of the notion of Metamorphic Relation (MR) and with the concepts of source and follow-up inputs. Our MRs consider two inputs and two outputs. Therefore, in the following definition we consider this situation (the extension to deal with more arguments is trivial).

Definition 4. Let $\mathcal{M} = (\mathcal{N}, \mathcal{W}_{\mathcal{N}})$ be an OSM map. A *metamorphic relation* R is a relation over two inputs, i_1 and i_2, and their corresponding outputs, $\mathcal{M}(i_1)$ and $\mathcal{M}(i_2)$. We can see R as a subset of a cartesian product. Specifically,

$$R \subseteq \mathcal{I}^2 \times (\mathcal{P}(\mathcal{N} \cup \mathcal{W}_{\mathcal{N}}))^2$$

We use the notation $R(i_1, i_2, \mathcal{M}(i_1), \mathcal{M}(i_2))$ to denote that $(i_1, i_2, \mathcal{M}(i_1), \mathcal{M}(i_2)) \in R$.

When we define an MR R, we will use the following pattern and notation. Let $i_1 = (T_1, C_1)$, $i_2 = (T_2, C_2)$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$
\begin{array}{|c|}
\hline
R(i_1, i_2, O_1, O_2)\\
\Updownarrow_{\mathbf{def}}\\
C(T_1, C_1, T_2, C_2, O_1, O_2)\\
\hline
\end{array}
$$

where C is a formula in first order logic without free variables.

Let $R(i_1, i_2, O_1, O_2)$ be an MR. We will say that i_1 is a *source input* and that i_2 is a *follow-up input*.

Fig. 3. Example of deadlock.

Note that source inputs are usually provided by testers while follow-up inputs should be automatically generated from source inputs and the definition of the MR.

5. Metamorphic relations: definition and implementation details

This section introduces the MRs that we propose in order to detect potential errors related to the information included in maps. The scope of application of our MRs is a specific map and their goal is to establish relations that must be fulfilled by the elements of the map. Our MRs assess the existence of connections of specific elements that appear in our maps. We focus on two sources of many errors: highways and roundabouts. We are interested in checking properties such as "a highway or a roundabout must have at least an entrance and an exit" or "there does not exist a highway in which different ways present different directions and, therefore, can lead to a deadlock".

5.1. `NoDeadlock`: *ways do not have deadlocks*

This MR aims at detecting ways in which two different segments of the way present opposite directions and there does not exist a *way out*. In Fig. 3 we show an example of this undesirable situation. In order to determine whether this situation happens, we look for ways with the same name and having the same final node. If we find a way with this feature, then we can initially assume that there is no exit, but there is an exception: we have to check whether there exists *another way* that traverses the former way that can be an exit.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, \{(name, \beta), (highway, *)\})$, $i_2 = (way, \{(name, !\beta), (highway, *)\})$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\texttt{NoDeadlock}(i_1, i_2, O_1, O_2)$$
$$\Updownarrow_{\textbf{def}}$$
$$\forall w_1, w_2 \in O_1 :$$
$$\left(\begin{array}{c} w_1 \neq w_2 \wedge last(w_1) = last(w_2) \wedge \\ (oneway, yes) \in tags(w_1) \cap tags(w_2) \\ \Downarrow \\ \exists w \in O_2 : last(w_1) \in set(w) \wedge last(w) \neq last(w_1) \end{array}\right)$$

Next, we show how inputs can be generated for this MR. The source input is a value, β_0, chosen by the tester according to some criteria. For example, it would be wise to choose a highway that is composed of many ways because we would increase the probability that some of the ways present an incorrect direction.

First, we let $i_0 = (way, \{(name, \beta_0), (highway, *)\})$. In addition, we consider the following follow-up input: $i'_0 = (way, \{(name, !\beta_0), (highway, *)\})$. We compute $O_0 = \mathcal{M}(i_0)$, $O'_0 = \mathcal{M}(i'_0)$ and check whether the following metamorphic relation holds:

$$\texttt{NoDeadlock}(i_0, i'_0, O_0, O'_0)$$

Fig. 4. Example of no exit way.

If the metamorphic relation does not hold, then we have found a failure and we stop the process. Otherwise, we will automatically generate additional inputs as follows. We consider the set $E = \{e | (name, \beta) \in tags(e) \wedge \beta \neq \beta_0 \wedge highway \in keys(e)\}$ and let $\lambda = |E|$. In other words, we are collecting the values *matching* $!\beta_0$ corresponding to names of highways. For each e_r, with $1 \leq r \leq \lambda$, let $(name, \beta_r) \in tags(e_r)$, $i_r = (way, \{(name, \beta_r), (highway, *)\})$, $i'_r = (way, \{(name, !\beta_r), (highway, *)\})$ and compute $O_r = \mathcal{M}(i_r)$ and $O'_r = \mathcal{M}(i'_r)$. We check the following λ metamorphic relations:

$$\texttt{NoDeadlock}(i_r, i'_r, O_r, O'_r)$$

If any of these applications of the MR does not hold, then we have found a failure and we stop the process. Otherwise, we can iterate the process by using the newly computed values having a different name.

5.2. `NoIsolatedWay`: *ways have entrances and/or exits*

This MR aims at detecting ways that do not intersect with any other way. In order to detect this situation, we look for ways such that none of their nodes appear in another way.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, \{(name, \beta), (highway, *)\})$, $i_2 = (way, \{(name, !\beta), (highway, *)\})$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\texttt{NoIsolatedWay}(i_1, i_2, O_1, O_2)$$
$$\Updownarrow_{\textbf{def}}$$
$$\exists w_1 \in O_1, w_2 \in O_2 : set(w_1) \cap set(w_2) \neq \emptyset$$

The generation of inputs is equal to the procedure described in Section 5.1 and, therefore, we omit it.

5.3. `ExitWay` *And* `EntranceWay`: *ways have at least one exit and one entrance*

The previous MR, `NoIsolatedWay`, focuses on ensuring that all the ways have either an exit or an entrance. In addition, we should ensure that they have at least an entrance and an exit. The next two metamorphic relations aim to check that all the ways have this condition.

Thus, given a certain highway, we should ensure that at least one node of the highway appears in some other way but the node is not the exit (resp. the entrance) of the way. In Fig. 4 we show an example of the situations that we would like to detect.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, \{(name, \beta), (highway, *)\})$, $i_2 = (way, \{(name, !\beta), (highway, *)\})$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\begin{array}{c} \texttt{ExitWay}(i_1, i_2, O_1, O_2) \\ \Updownarrow_{\textbf{def}} \\ \exists w_1 \in O_1, w_2 \in O_2, n \in set(w_1) \cap set(w_2) : \\ \left(\begin{array}{c} (noexit, yes) \notin tags(w_1) \\ \Downarrow \\ n \neq last(w_2) \vee (oneway, yes) \notin tags(w_2) \end{array} \right) \end{array}$$

$$\begin{array}{c} \texttt{EntranceWay}(i_1, i_2, O_1, O_2) \\ \Updownarrow_{\textbf{def}} \\ \exists w_1 \in O_1, w_2 \in O_2, n \in set(w_1) \cap set(w_2) : \\ n \neq first(w_2) \vee (oneway, yes) \notin tags(w_2) \end{array}$$

In this case, as in the previous one, the generation of inputs is equal to the procedure described in Section 5.1 and, therefore, we omit it.

5.4. `ExitRAbout` *And* `EntRAbout`*: connected roundabouts*

Next we would like to ensure that our map does not have roundabouts without exits or entrances. The former happens when at least one of the nodes in the ways that conform the roundabout correspond to the last node of another way. If we find a way that presents this feature, then we can assume that there is an entrance. The latter happens when there exists a way whose first node corresponds to any of the nodes that appear in the roundabout. We propose two different MRs for detecting these anomalies.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, \{(name, \beta), (highway, *)\})$, $i_2 = (way, (highway, *))$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\begin{array}{c} \texttt{ExitRAbout}(i_1, i_2, O_1, O_2) \\ \Updownarrow_{\textbf{def}} \\ \exists w_1 \in O_1, w_2 \in O_2 : (junction, roundabout) \in tags(w_1) \\ \wedge \\ (junction, roundabout) \notin tags(w_2) \\ \wedge \\ first(w_2) \in set(w_1) \end{array}$$

$$\begin{array}{c} \texttt{EntRAbout}(i_1, i_2, O_1, O_2) \\ \Updownarrow_{\textbf{def}} \\ \exists w_1 \in O_1, w_2 \in O_2 : (junction, roundabout) \in tags(w_1) \\ \wedge \\ (junction, roundabout) \notin tags(w_2) \\ \wedge \\ last(w_2) \in set(w_1) \end{array}$$

The fact that a roundabout and a way reaching it or outgoing from it can have the same name makes necessary to restrict the ways selected by the input i_2 to those that do not include the $(junction, roundabout)$ tag. Otherwise, the MR always holds.

In contrast to the previous MRs, we cannot automatically generate an unlimited number of source and follow-up inputs for this MR because the role of the tester is basic to consider only *sensible* values. This is not an unfamiliar situation in MT. For example, if we consider the $\sin(x) = \sin(x + 2 \cdot \pi)$ MR, then the second argument of the relation will be automatically produced from the first one but the tester must select *good* values of x. We have exactly the same situation here. Therefore, the tester has to select a value β_0 of interest for being used as *source input*. This value must correspond to the name of a way that has associated the tag $(junction, roundabout)$ and a tag with the key $highway$. Then, we will consider the inputs $i_0 = (way, \{(name, \beta_0), (highway, *)\})$ and $i'_0 = (way, (highway, *))$, and will compute $O_0 = \mathcal{M}(i_0)$ and $O'_0 = \mathcal{M}(i'_0)$. Finally, we will check whether the following metamorphic relations hold:

$$\texttt{ExitRAbout}(i_0, i'_0, O_0, O'_0)$$

$$\texttt{EntRAbout}(i_0, i'_0, O_0, O'_0)$$

If any of them does not hold, then we have found a failure. Otherwise, we will look for another appropriate value and iterate the process.

5.5. `Connected`*: connected ways*

Unless a way has associated the tag $(noexit, yes)$, it should be connected to another way. Connected ways share the last node with another way. The objective of this MR is to ensure that our map does not have unconnected ways.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, \{(name, \beta), (highway, *)\})$, $i_2 = (way, \{(name, !\beta), (highway, *)\})$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\begin{array}{c} \texttt{Connected}(i_1, i_2, O_1, O_2) \\ \Updownarrow_{\textbf{def}} \\ \forall w_1 \in O_1 : \\ \left(\begin{array}{c} (noexit, yes) \notin tags(w_1) \\ \Downarrow \\ \exists w \in O_1 \cup O_2 : w \neq w_1 \wedge last(w_1) \in set(w) \end{array} \right) \end{array}$$

The generation of inputs is equal to the procedure described in Section 5.1. The only aspect that we must point out is that, in this case, the tester should select a value β_0 for the source input corresponding to the name of a highway that does not have the tag $(noexit, yes)$. In the same way, the tester should not consider source inputs that do not fulfill this condition.

5.6. `AreaNoInt`*: area not intersected*

Closed ways can be used to represent buildings and landuse places, among others. Some closed ways have the tag $(area, yes)$. The next MR will help us to check that closed ways (in both cases) are not intersected by other ways, which makes no sense. The relation checks whether any of the segments defined by consecutive nodes that constituted the closed way is traversed by a segment belonging to another way. There is an exception: if the way is labeled with the tags railway, highway or waterway, then it can be crossed by another way. It is worth pointing out that the goal of this MR is to detect *intersections*, not *overlap* of ways. The possible overlaps of these elements are considered in the next MR.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, (name, \beta))$, $i_2 = (way, (name, !\beta))$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\texttt{AreaNoInt}(i_1, i_2, O_1, O_2)$$

$$\Updownarrow_{\textbf{def}}$$

$$\begin{array}{c}\forall w_1 \in O_1, n_1, n_2 \in set(w_1),\\ w_2 \in O_2, n'_1, n'_2 \in set(w_2) :\end{array}$$

$$\left(\begin{array}{c}\left(\begin{array}{c}\left(\begin{array}{c}(area, yes) \in tags(w_1)\\ \vee\\ \left(\begin{array}{c}keys(w_1)\\ \cap\\ \{building, landuse\}\end{array}\right) \neq \emptyset\end{array}\right)\\ \wedge\\ \left(\begin{array}{c}keys(w_2)\\ \cap\\ \{highway, waterway, railway\}\end{array}\right) = \emptyset\\ \wedge\\ n_1 = prev(w_1, n_2) \wedge n'_1 = prev(w_2, n'_2)\end{array}\right)\\ \Downarrow\\ line(n_1, n_2) \cap line(n'_1, n'_2) = \emptyset\\ \vee\\ n_1, n_2 \in line(n'_1, n'_2)\\ \vee\\ \left(\begin{array}{c}line(n_1, n_2) \cap line(n'_1, n'_2) = \{p\}\\ \wedge\\ (p \notin seg(n_1, n_2) \vee p \notin seg(n'_1, n'_2))\end{array}\right)\end{array}\right)$$

We have a similar situation to the one described in Section 5.4 with respect to the generation of inputs for this MR. It is not possible to automatically produce an unlimited number of inputs because the values that the tester has to choose must be *appropriate* for the testing process. In this case, the tester has to select a value β_0 that must be associated to the key *name* of a way. In order to speed-up the process, the corresponding element should have either the tag $(area, yes)$ or the keys $building$ or $landuse$. Then, this value will be used as source input, consider $i_0 = (way, (name, \beta_0))$, use $i'_0 = (way, (name, !\beta_0))$ as follow-up input, compute $O_0 = \mathcal{M}(i_0)$ and $O'_0 = \mathcal{M}(i'_0)$, and check whether the following application of the MR holds:

$$\texttt{AreaNoInt}(i_0, i'_0, O_0, O'_0)$$

If it does not hold, then we have found a failure. As in the previous MR, the tester should take into account the condition imposed in the MR, that is, the values selected for the source and the follow-up inputs should correspond to elements that do not have associated tags with any of the keys $highway$, $waterway$ and $railway$.

5.7. `NoOverlap`: *no overlapping highways or buildings*

Our last MR ensures that highways and buildings do not overlap, that is, they do not share any segment.

Given a string β, denoting the name of a way to be analyzed, let $i_1 = (way, (name, \beta))$, $i_2 = (way, (name, !\beta))$, $O_1 = \mathcal{M}(i_1)$ and $O_2 = \mathcal{M}(i_2)$.

$$\texttt{NoOverlap}(i_1, i_2, O_1, O_2)$$

$$\Updownarrow_{\textbf{def}}$$

$$\begin{array}{c}\forall w_1 \in O_1, n_1, n_2 \in set(w_1),\\ w_2 \in O_2, n'_1, n'_2 \in set(w_2) :\end{array}$$

$$\left(\begin{array}{c}keys(w_1) \cap \{highway, building\} \neq \emptyset\\ \wedge\\ keys(w_2) \cap \{highway, building\} \neq \emptyset\\ \wedge\\ n_1 = prev(w_1, n_2) \wedge n'_1 = prev(w_2, n'_2)\\ \Downarrow\\ \neg\ overlap(seg(n_1, n_2), seg(n'_1, n'_2))\end{array}\right)$$

The generation of inputs follows, essentially, the same procedure described in Section 5.1 and, therefore, we omit. Nevertheless, we have to point out that it will be convenient that the values selected by the tester for the source and follow-up inputs do have an associated tag that includes either $highway$ or $building$ as key.

5.8. Implementation details

We have implemented the proposed MRs and applied them on real OSM maps (our experiments are reported in the next section). Since OSM maps can be exported in XML format, we have used the XML Database Query Language XQuery [42,43] to implement our framework. XQuery is a functional language based on FLWOR ("For, Let, Where, Order by, Return") expressions, and using the XPath language to select nodes of XML trees. The fact that XQuery is a declarative programming language strongly facilitates the *translation* of our MRs into this language: it is almost direct.

Thanks to higher order capabilities of XQuery, the application of inputs can be defined by an XQuery higher order function. For instance, if we have an input $i = (T, C)$, then the application of this input to a map, $\mathcal{M}(i)$, is defined in XQuery as:

```
declare function mt:M($T,$C)
{
filter($map/osm/*,
    function($e){mt:type($e,$T) and $C($e)})
};
```

Inputs of MRs are mapped to XQuery Boolean conditions, using XQuery "some" statements and XPath expressions. For example, the *translation* of the input condition $C = \{(name, \beta), (highway, *)\}$ is defined as:

```
some $z in $x/tag satisfies $z[@k="name"and @v=$beta]
    and (some $z in $x/tag satisfies $z[@k="highway"])
```

where x/tag stands for the tags of x and $(@k, @v)$ for the (key, value) pairs of a tag. MRs are mapped to XQuery Boolean conditions using XQuery "some" and "every" statements, as well as union/intersection operators. For example, the `NoIsolatedWay` MR can be defined as:

```
some $w1 in $O1
satisfies
some $w2 in $O2
satisfies
  some $n in $w1/nd
  satisfies
  some $m in $w2/nd
  satisfies
    $n/@ref=$m/@ref
```

where $w1/nd$ (resp. $w2/nd$) represents the nodes of $w1$ (resp. $w2$) and $n/@ref$ (resp. $m/@ref$) the node id of n (resp. m). The defined functions become XQuery functions, except *prev* which can be defined as XPath axes. For example, the `NoOverlap` MR can be defined as:

```
every $w1 in $O1
satisfies
every $w2 in $O2
satisfies
    every $n2 in $w1/nd
    satisfies
    every $n2p in $w2/nd
    satisfies
     let $n1 :=
       ($n2/preceding-sibling::node())[last()]
     let $n1p :=
       ($n2p/preceding-sibling::node())[last()]
     return
     not(mt:overlapSegment(mt:Node($n1),mt:Node($n2),
                  mt:Node($n1p),mt:Node($n2p)))
```

The implementation uses the XQuery language for storing and accessing XML representation of OSM maps. Fortunately, XML documents are automatically indexed by XQuery and thus efficient querying and accessing are guaranteed to sequences of items of OSM maps. Inputs

are represented as sequences of XML items and filtering/elimination is achieved by an XQuery query.

All the code used to perform the experiments, in particular the implementation of our MRs, can be found at https://github.com/jalmenUAL/MT-OSM.

6. Empirical evaluation

In this section we describe the experiments used to assess the ability of our framework to detect errors in OSM. First, we pose our research questions. Later on, and based on the results of the experiments, we provide answers to these questions and analyze potential threats to the validity of the results.

6.1. Research questions

We consider three research questions. The first one concerns scalability. Even if our proposal were able to find errors, its usefulness would be minor if it cannot cope with medium/big size areas of OSM.

Research Question 1. *Does our framework scale well?*

The second one concerns the ability to find errors.

Research Question 2. *Is our framework an effective way to find errors in maps?*

Finally, our third research question is related to the usefulness of our approach with respect to alternative approaches.

Research Question 3. *How does our framework compare to other approaches?*

6.2. Experiments

We have conducted experiments in four cities: Madrid, London, Sao Paulo and Nairobi. For each MR, we were able to find a place in all the cities where the MR does not hold. In our experiments, we excluded cases like the "propose" and "closed" tags for roads that are gazetted but not yet built. Typically, the violation of the MR can be caused by more than one single OSM item. Figs. 5, 6 and 7 show an example of the contravention of each MR in each city.

The violation of `NoDeadlock` is caused by streets with wrong opposite directions (see Fig. 5-(a), (d) and (g)) and loops (see Fig. 5-(j)).

With regard to `ExitRAbout` and `EntRAbout`, in some cases a roundabout is defined by segments, instead of a unique way, which causes that one of the segments does not have either exit or entrance. This type of error has been detected, among others, in the situations shown in Fig. 5-(e), (f) and (i). In some other cases, the roundabout and the highway crossing it overlap. Moreover, in these cases the roundabout is unnamed. This reveals again an error since the highway should be defined by two segments linked to the corresponding roundabout. This type of error is shown in Fig. 5-(b) and (c). Another interesting case is given in Fig. 5-(h). Here, there is a deadlock in a candidate exit. This causes an `ExitRAbout` error. Finally, there are two additional types of errors detected by these MRs. The situation shown in Fig. 5-(k) represents a highway linked to a roundabout without exit (i.e. the roundabout becomes a loop). Besides, in Fig. 5-(l) we can see a roundabout represented by one node which is also a node of the highway. Therefore, we have a roundabout without entrance.

The violation of the `Connected` MR is mainly produced by missing tags "noexit" and "yes" (see Fig. 6-(a), (d), (g) and (j)). Besides, the violation of `NoIsolatedWay` is due to two main situations. The first one appears when we have completely disconnected ways. This can be seen in Fig. 6-(b), (e) and (h). In the second situation, the way is isolated because the way crossing it is unnamed Fig. 6-(k). We have often detected this situation in residential areas.

`ExitWay` and `EntranceWay` are able to find errors of different nature. Figs. 6-(c) and 7-(a) show the lack of accuracy in geometries. Here, there exists an entrance (resp. exit) for the way but there is no way to get in (resp. exit) because the direction of the candidate way is opposite to the entrance (resp. exit) direction. Figs. 6-(f) and 7-(g) show, again, a lack of accuracy in geometries, in which two nodes represent the same geographic point. Thus, we detect unconnected ways from the OSM rules point of view. Finally, Figs. 6-(i) and (l) and 7-(d) and (j) show ways with an entrance (resp. exit) but no exit (resp. entrance).

Most errors detected by `AreaNoInt` and `NoOverlap` were due to the lack of accuracy of geometries producing intersection and overlapping. For instance, and with respect to intersection, Fig. 7-(b) shows an intersection of grassland with a footway. In Fig. 7-(e) there is an intersection point among the building Crutched Friars and London Fenchurch Street. In Fig. 7-(h), a residential landuse includes an intersection point with a building. Fig. 7-(k) shows an intersection point between two buildings. With respect to overlap, most errors happen with buildings (see Fig. 7-(c), (f) and (i)). Finally, Fig. 7-(l) shows an overlap between a building and an area representing a forest.

We would like to conclude this section with a small discussion on the king of errors that we are able to detect. It is important to emphasize that the errors shown in Table 4 are categorized as *high severity* in the scale proposed by *Osmose*. High severity means a real problem on the map that needs to be solved by the OSM community using the tools available for it. For example, defects detected by MRs such as `NoDeadLock` and `ExitWay` can give rise to potential errors in route planners that use OSM as a base map. In the case of MRs related to areas, that is, `AreNoInt` and `NoOverlap`, the targeted errors are of high severity because the real world objects are incorrectly represented in OSM.

Finally, let us mention that some of the reported errors can be false positives. That is, even though the geometry is correct, a mistake on the labeling produces the violation of a certain MR. This happens, for instance, in absence of a $(noexit, yes)$ tag, which could lead to the violation of `Connected`, as well as in presence of a $(oneway, yes)$, which could lead to the violation of `ExitWay` and `EntranceWay`. Other problematic situations can be produced when labeling is similarly involved.

6.3. Answers to the research questions

We think that the previous results allow us to provide a positive answer to our Research Question 2: our approach is effective to find errors in OSM.

Typically, the tools analyzing OSM are applied in small/medium size areas. Big cities areas should be split into small/medium bounding boxes to be studied. In order to estimate how big the bounding boxes should be, we ran our MRs with several areas of the same city (Madrid). In Table 2 we present the coordinates of the chosen area, the size of the corresponding map and the number of elements included in the map (nodes and ways). The results concerning running time and number of errors detected by each MR, in the studied area, are shown in Table 3. As expected, some MRs require more time than others, but most of them run in reasonable time for sizes smaller than 150 megabytes. Actually, we are able to conclude the analysis of even the bigger maps for all of the MRs with the exception of two applications of the `NoOverlap` MR. Therefore, we can affirmatively answer to our Research Question 1: our approach scales well.

Finally, in order to provide an answer to Research Question 3, we need to compare the obtained results with other alternative approaches. In order to perform this comparison, we decided to rely on the most widely used OSM error checker *Osmose*. Thus, we can analyze whether the problems found by our framework have been reported before. Table 4 shows our results concerning this comparison. For each of the items previously enumerated, we give links to the *Osmose* tool, OSM map and OSM API. Interestingly enough, we found that 65% of the errors found by our MRs have not (still) been reported by *Osmose*. Therefore, we can claim that our framework can find errors that have not been previously found by other tools.

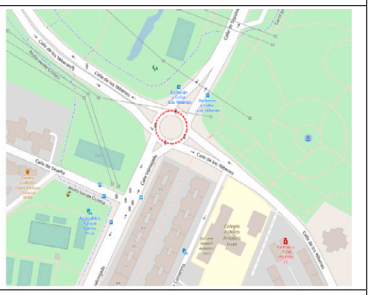
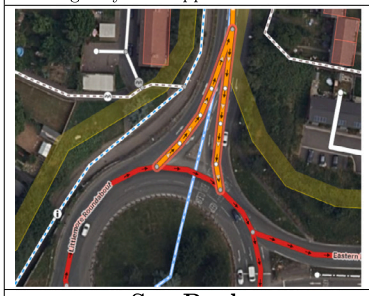
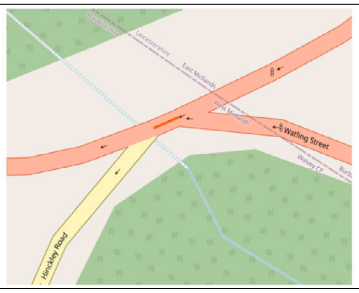
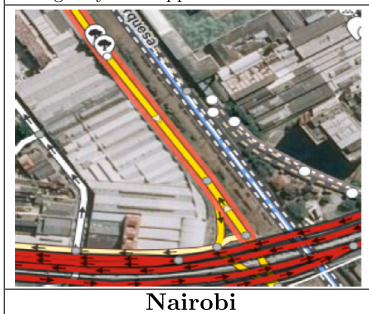
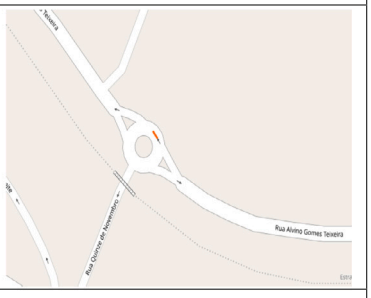
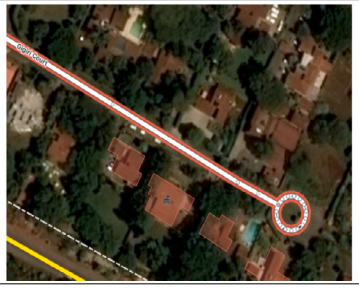
NoDeadlock	ExitRAbout	EntRAbout
Madrid		
(a) 15219050-Calle de Almería Highway with opposite directions	(b) 760922920-Unnamed	(c) 27879095-Unnamed
		
London		
(d) 2444219-Oxford Road Highway with opposite directions	(e) 698957939-Unnamed	(f) 788026956-Unnamed
		
Sao Paulo		
(g) 75416395-Rua Palmorino Mônaco Highway with opposite direction	(h) 668343924-Unnamed	(i) 810836495-Unnamed
		
Nairobi		
(j) 296151761-Moi Avenue Highway with loop	(k) 25063949-Gigiri Court	(l) 1390094943-Unnamed
		

Fig. 5. Errors detected by MRs I.

Table 2
Scalability datasets.

Bounding Box	Size (MB)	Elements	Nodes	Ways
−3.67064, 40.42702, −3.65900, 40.43178	2.9	8743	7587	1156
−3-67870, 40.42300, −3.65540, 40.43260	8.6	33500	28718	4782
−3.70090, 40.41370, −3.65430, 40.43270	15.9	109712	95431	14281
−3.72420, 40.40020, −3.63110, 40.44220	59.4	427036	369592	57444
−3.77070, 40.38510, −3.58450, 40.46130	123.5	1000720	859882	140838
−3.86380, 40.34710, −3.49120, 40.49940	251.9	2160991	1851862	309129
−4.04980, 40.27060, −3.30480, 40.57540	447.8	3959529	3408487	551042
−4.42200, 40.11800, −2.93200, 40.72750	769.8	6088426	5318702	769724

Fig. 6. Errors detected by MRs II.

Table 3
Scalability tests.

	NoDeadLock		NoIsolatedWay		ExitRAbout		EntRAbout		Connected		ExitWay		EntranceWay		AreaNoInt		NoOverlap	
Size (MB)	RT (ms)	#	RT (ms)	#	RT (ms)	#	RT (ms)	#	RT (ms)	#	RT (ms)	#	RT (ms)	#	RT (ms)	#	RT (ms)	#
2.9	20.76	1	1006.65	1	170.12	0	195.09	0	539.44	1	1495.24	1	1565.93	1	1168	1	15805.55	1
8.6	35.65	3	11231.67	5	540.45	0	499.66	0	714.09	2	8345.82	4	4721.65	2	7857.15	1	170906.75	2
15.9	782.02	7	24783.03	12	1549.42	4	785.09	3	1388.65	4	11742.67	9	7645.54	4	11040.57	2	380341.85	4
59.4	2867.03	14	39847.43	23	6250.23	12	3325.54	6	5858.54	14	19200.87	13	15439.64	14	19463.95	9	880982.34	8
123.5	6354.56	31	78546.78	44	11867.34	20	5964.32	10	37755.45	53	375118.75	42	244880.92	37	24580.36	16	1389002.23	11
251.9	18300.56	49	189819.45	63	20980.66	34	15386.6	22	95594.67	124	547775.39	79	420556.82	62	31450.42	18	2340556.92	14
447.8	30356.87	63	303904.94	86	34890.43	49	27327.8	35	167094.8	187	884329.87	95	599881.81	82	56850.52	21	Stack overflow	
769.8	45565.32	69	530469.25	103	64811.24	54	45773.54	44	267025.65	230	1208743.45	135	898453.28	109	77051.78	28	Stack overflow	

Fig. 7. Errors detected by MRs III.

6.4. Threats to the validity of the results

Next we discuss the threats to the validity of the results of the experiments reported in the previous section. We distinguish three categories.

- Threats to *internal validity* consider uncontrolled factors that might be responsible for the obtained results.
- Threats to *external validity* consider whether the conclusions of our experiments can be generalized and adapted to other situations.
- Threats to *construct validity* consider whether we are working with properties of interest.

Concerning threats to internal validity, our main worry was related to the possible faults in the developed code. In order to reduce the impact of this threat, we thoroughly tested the code implementing our MRs, using carefully constructed examples for which we could (manually, if needed) check the results. In order to reach the high level of automation that our framework provides, it is important to ensure that the (automatic) generation of follow-up inputs is sound. Again, we meticulously checked that follow-up inputs were uninterruptedly produced, with a special care to avoid repetitions. Note that if repeated follow-up inputs are not appropriately discarded, then we might uselessly check certain areas of a map while others might not be analyzed. Another important concern is related to the definition of our MRs. Obviously, the ability of our framework to detect errors in OSM is directly correlated with the selection of MRs and, consequently, the results may be different if other MRs were used instead. Our MRs were

carefully designed by the authors and discussed with colleagues also working on MT.

We considered two main threats to external validity. The first one is whether the ability of our MRs to find errors would be similar in different situations. For our experiments, we chose four different cities in three continents. The rationale is that if our framework is able to provide a good performance when applied to these heterogeneous *case studies*, then we can diminish the impact of good/bad contributors in the ability of our framework to find errors. We think that these cities are indeed representative of OSM and, therefore, we expect that the obtained results for measuring the effectiveness of the MRs will be similar in other cities. The second threat considers whether our framework can be adapted to deal with other notations to represent maps. Here, we have a mixed response. As long as the alternative representation would allow us to access its data, we should be able to deal with it. Unfortunately, this is not currently possible for commercial applications like Bing,[15] Google[16] and Wikimapia[17] or Yandex[18] maps. If a bigger access is provided in the future, then our framework should be able to deal with them because these systems have similar representations for geometries (i.e., *Point*, *Linestring*, *Polygon*, etc.,) to the ones used in OSM and tagging systems, to a greater or lesser extent, are also available. The Bing tagging system is limited to *title* and *subTitle*. However, the Google tagging system is richer than Bing because it enables the description of objects with a *feature* attribute in which information such as *poi.attraction*, *landscape.natural*, *road.highway*, etc., can be specified. WikiMapia allows any contributor to add a "tag" (placemark) to any location providing a default language, title, description and one or more categories. Finally, Yandex has a tagging system similar to OSM.

In order to consider threats to construct validity, we have to evaluate the fault detection ability of the generated input values and the time needed to both obtain them and apply them to a certain area. In order to diminish this threat, in our experiments we used different cities and split them in areas of different sizes (from small to very big). Undetected errors in the definition of our proposed MRs or in our algorithm for generating follow-up inputs are also a threat to construct validity. We control this threat by executing a wide range of experiments that were able to completely cover the analyzed areas with the exception of two big areas for one MR. Finally, the proposed MR framework uses the language XQuery and the compliant XQuery 3.1 processor BaseX.[19] This establishes a dependence of our framework on external programs, as well as user skills on software development with XQuery for building code for new MRs.

7. Conclusions and future work

We have presented an MT approach to validate the information included in OSM maps. Specifically, we were interested in erroneous definitions of geometries. Our previous work on OSM was useful, as a starting point, because we had experience on analyzing the quality of OSM. However, the framework presented in this paper represents a big step forward because it provides a structured methodology, with a formal mathematical underlying basis. It is important to emphasize that our framework is fully implemented (in particular, the application of inputs to a map, in order to obtain the elements matching the input, and all the presented MRs). Therefore, our framework can be used to analyze other cities. Actually, users can either use our MRs or use the ones that they define. We have experimental evidence to show the usefulness of our MRs and of our framework: we have found several errors in all the cities that we analyzed.

[15] https://www.bing.com/maps/.
[16] https://www.google.com/maps/.
[17] https://wikimapia.org/.
[18] https://yandex.com/maps/.
[19] https://basex.org/.

Table 4
Osmose errors (including clickable links to the errors in different formats).

Item	OSMOSE	Rep.	Detected Problem	Scale	OSM	XML
Fig. 5-(a)	Link	**No**	One way inaccessible	High	Link	Link
Fig. 5-(b)	Link	Yes	Broken highway cont.	High	Link	Link
Fig. 5-(c)	Link	Yes	Broken highway cont.	High	Link	Link
Fig. 5-(d)	Link	Yes	Opposite lane in the same way of a oneway	High	Link	Link
Fig. 5-(e)	Link	Yes	No Needed Label - junction=roundabout	High	Link	Link
Fig. 5-(f)	Link	Yes	No Needed Label - junction=roundabout	High	Link	Link
Fig. 5-(g)	Link	**No**	One way inaccessible	High	Link	Link
Fig. 5-(h)	Link	**No**	Broken highway cont.	High	Link	Link
Fig. 5-(i)	Link	Yes	No Needed Label - junction=roundabout	High	Link	Link
Fig. 5-(j)	Link	**No**	Highway with loop	Normal	Link	Link
Fig. 5-(k)	Link	**No**	Broken highway cont.	High	Link	Link
Fig. 5-(l)	Link	Yes	No Needed Label - junction=roundabout	High	Link	Link
Fig. 6-(a)	Link	Yes	Almost junction, join or use noexit tag	High	Link	Link
Fig. 6-(b)	Link	Yes	Highway with missing, wrong or abbreviated street address	High	Link	Link
Fig. 6-(c)	Link	Yes	One way inaccessible or no parking or parking entrance	High	Link	Link
Fig. 6-(d)	Link	**No**	Almost junction, join or use noexit tag	High	Link	Link
Fig. 6-(e)	Link	**No**	Almost junction, join or use noexit tag	High	Link	Link
Fig. 6-(f)	Link	Yes	Road above ground and no bridge	High	Link	Link
Fig. 6-(g)	Link	**No**	Almost junction, join or use noexit tag	High	Link	Link
Fig. 6-(h)	Link	Yes	Highway with missing, wrong or abbreviated street address	High	Link	Link
Fig. 6-(i)	Link	Yes	Intersection of unrelated roads and objects	High	Link	Link
Fig. 6-(j)	Link	**No**	Almost junction, join or use noexit tag	High	Link	Link
Fig. 6-(k)	Link	**No**	Almost junction, join or use noexit tag	High	Link	Link
Fig. 6-(l)	Link	**No**	Highway with missing, wrong or abbreviated street address	High	Link	Link
Fig. 7-(a)	Link	Yes	Suspicious tag combination	Normal	Link	Link
Fig. 7-(b)	Link	Yes	Area intersects with area	High	Link	Link
Fig. 7-(c)	Link	Yes	Large building intersection	High	Link	Link
Fig. 7-(d)	Link	Yes	One way inaccessible or no parking or parking entrance	High	Link	Link

(*continued on next page*)

Table 4 (*continued*).

Item	OSMOSE	Rep.	Detected Problem	Scale	OSM	XML
Fig. 7-(e)	Link	Yes	Suspicious key ending with a number	Normal	Link	Link
Fig. 7-(f)	Link	**No**	Building intersection	High	Link	Link
Fig. 7-(g)	Link	Yes	Disconnected road	High	Link	Link
Fig. 7-(h)	Link	**No**	Building inters. area	High	Link	Link
Fig. 7-(i)	Link	Yes	Large building intersection	High	Link	Link
Fig. 7-(j)	Link	**No**	Disconnected road	High	Link	Link
Fig. 7-(k)	Link	Yes	Building intersection	High	Link	Link
Fig. 7-(l)	Link	Yes	Large building intersection	High	Link	Link

We are considering several lines for future work. We would like to analyze whether other groups of MRs, targeting other sources of errors in OSM maps, can be defined. Although our framework has been fully implemented, and it is available to be used with other OSM maps, we would like to implement a GUI on top of our code so that users can apply our framework in a more user-friendly way. The user should be able to choose a map, the desired MRs, provide source inputs (potentially suggested by the GUI) and retrieve a list of found errors. In order to further evaluate the usefulness of our approach, we would like to adapt current mutation testing techniques [44,45] to automatically produce *mutated* portions of maps and check whether our MRs are able to detect the induced faults. In a more theoretical line of work, we will try to adapt our previous work on passive testing of (distributed) systems with multiple users [46–48] to *passively* test the interaction of contributors with a specific OSM.

CRediT authorship contribution statement

Jesús M. Almendros-Jiménez: Conceptualization, Software, Methodology, Writing - original draft, Writing - review & editing, Funding acquisition. **Antonio Becerra-Terón:** Conceptualization, Software, Validation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Mercedes G. Merayo:** Conceptualization, Methodology, Formal analysis, Writing - original draft, Writing - review & editing, Funding acquisition. **Manuel Núñez:** Conceptualization, Methodology, Formal analysis, Writing - original draft, Writing - review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, third ed., John Wiley & Sons, 2011.

[2] P. Ammann, J. Offutt, Introduction To Software Testing, second ed., Cambridge University Press, 2017.

[3] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, The oracle problem in software testing: A survey, IEEE Trans. Softw. Eng. 41 (5) (2015) 507–525.

[4] E.J. Weyuker, On testing non-testable programs, Comput. J. 25 (4) (1982) 465–470.

[5] T.Y. Chen, S.C. Cheung, S.M. Yiu, Metamorphic Testing: a New Approach for Generating Next Test Cases, Tech. Rep., (HKUST-CS98-01) Department of Computer Science, Hong Kong University of Science and Technology, 1998.

[6] T.Y. Chen, T.H. Tse, Z.Q. Zhou, Fault-based testing without the need of oracles, Inf. Softw. Technol. 45 (1) (2003) 1–9.

[7] S. Segura, D. Towey, Z.Q. Zhou, T.Y. Chen, Metamorphic testing: Testing the untestable, IEEE Softw. 37 (3) (2020) 46–53.

[8] H. Liu, F.-C. Kuo, D. Towey, T.Y. Chen, How effectively does metamorphic testing alleviate the oracle problem? IEEE Trans. Softw. Eng. 40 (1) (2014) 4–22.

[9] P. Neis, A. Zipf, Analyzing the contributor activity of a volunteered geographic information project–the case of OpenStreetMap, ISPRS Int. J. Geo-Inf. 1 (2) (2012) 146–165.

[10] P. Mooney, P. Corcoran, Has openstreetmap a role in digital earth applications? Int. J. Digit. Earth 7 (7) (2014) 534–553.

[11] J. Jokar Arsanjani, A. Zipf, P. Mooney, M. Helbich (Eds.), OpenStreetMap in GIScience: experiences, research and applications, in: Lecture Notes in Geoinformation and Cartography, Springer, 2015.

[12] A. Núñez, P.C. Cañizares, M. Núñez, R.M. Hierons, TEA-Cloud: A formal framework for testing cloud computing systems, IEEE Trans. Reliab. 70 (1) (2021) 261–284.

[13] C. Lidbury, A. Lascu, N. Chong, A.F. Donaldson, Many-core compiler fuzzing, in: 36th ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI'15, ACM, 2015, pp. 65–76.

[14] T.Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, Z.Q. Zhou, Metamorphic testing for cybersecurity, Computer 49 (6) (2016) 48–55.

[15] S. Pugh, M.S. Raunak, D.R Kuhn, R. Kacker, Systematic testing of post-quantum cryptographic implementations using metamorphic testing, in: 4th Int. Workshop on Metamorphic Testing, MET'19, IEEE, 2011, pp. 2–8.

[16] Z.Q. Zhou, S. Xiang, T.Y. Chen, Metamorphic testing for software quality assessment: A study of search engines, IEEE Trans. Softw. Eng. 42 (3) (2016) 264–284.

[17] P.C. Cañizares, A. Núñez, J. de Lara, An expert system for checking the correctness of memory systems using simulation and metamorphic testing, Expert Syst. Appl. 132 (2019) 44–62.

[18] M. Olsen, M. Raunak, Increasing validity of simulation models through metamorphic testing, IEEE Trans. Reliab. 68 (1) (2019) 91–108.

[19] B. Yan, B. Yecies, Z.Q. Zhou, Metamorphic relations for data validation: A case study of translated text messages, in: IEEE/ACM 4th Int. Workshop on Metamorphic Testing, MET'19, IEEE, 2019, pp. 70–75.

[20] Z.Q. Zhou, L. Sun, T.Y. Chen, D. Towey, Metamorphic relations for enhancing system understanding and use, IEEE Trans. Softw. Eng. 46 (10) (2020) 1120–1154.

[21] S. Segura, G. Fraser, A.B. Sánchez, A. Ruiz-Cortés, A survey on metamorphic testing, IEEE Trans. Softw. Eng. 42 (9) (2016) 805–824.

[22] T.Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T.H. Tse, Z.Q. Zhou, Metamorphic testing: A review of challenges and opportunities, ACM Comput. Surv. 51 (1) (2018) 4:1–4:27.

[23] J. Brown, Z.Q. Zhou, Y.-W. Chow, Metamorphic testing of navigation software: A pilot study with google maps, in: 51st Hawaii Int. Conf. on System Sciences, HICSS'18, ScholarSpace / AIS Electronic Library (AISeL), 2018, pp. 1–10.

[24] J. Brown, Z.Q. Zhou, Y.-W. Chow, Metamorphic testing of mapping software, in: Towards Integrated Web, Mobile, and IoT Technology, LNBIP 347, Springer, 2019, pp. 1–20.

[25] Z. Hui, S. Huang, C. Chua, T.Y. Chen, Semiautomated metamorphic testing approach for geographic information systems: An empirical study, IEEE Trans. Reliab. 69 (2) (2020) 657–673.

[26] L.C. Degrossi, J. Porto de Albuquerque, R. dos Santos Rocha, A. Zipf, A taxonomy of quality assessment methods for volunteered and crowdsourced geographic information, Trans. GIS 22 (2) (2018) 542–560.

[27] A. Basiri, M. Haklay, G. Foody, P. Mooney, Crowdsourced geospatial data quality: challenges and future directions, Int. J. Geogr. Inf. Sci. 33 (8) (2019) 1588–1593.

[28] H. Zhang, J. Malczewski, Quality evaluation of volunteered geographic information: The case of openstreetmap, in: Crowdsourcing: Concepts, Methodologies, Tools, and Applications, IGI Global, 2019, pp. 1173–1201.

[29] S.S. Sehra, J. Singh, H.S. Rai, S.S. Anand, Extending processing toolbox for assessing the logical consistency of openstreetmap data, Trans. GIS 24 (1) (2020) 44–71.

[30] M. Haklay, S. Basiouka, V. Antoniou, A. Ather, How many volunteers does it take to map an area well? The validity of Linus law to volunteered geographic information, Cartogr. J. 47 (4) (2010) 315–322.

[31] P. Mooney, P. Corcoran, Characteristics of heavily edited objects in OpenStreetMap, Future Internet 4 (1) (2012) 285–305.

[32] C. Keßler, R.T.A. De Groot, Trust as a proxy measure for the quality of volunteered geographic information in the case of OpenStreetMap, in: Geographic Information Science At the Heart of Europe, Springer, 2013, pp. 21–37.

[33] C. Barron, P. Neis, A. Zipf, A comprehensive framework for intrinsic OpenStreetMap quality analysis, Trans. GIS 18 (6) (2014) 877–895.

[34] A. Vandecasteele, R. Devillers, Improving volunteered geographic information quality using a tag recommender system: the case of OpenStreetMap, in: OpenStreetMap in GIScience, Springer, 2015, pp. 59–80.

[35] S.S. Sehra, J. Singh, H.S. Rai, Assessing OpenStreetMap data using intrinsic quality indicators: an extension to the QGIS processing toolbox, Future Internet 9 (2) (2017) 15.

[36] P. Fogliaroni, F. D'Antonio, E. Clementini, Data trustworthiness and user reputation as indicators of VGI quality, Geo-spatial Inf. Sci. 21 (3) (2018) 213–233.

[37] A. Nasiri, R. Ali Abbaspour, A. Chehreghan, J. Jokar Arsanjani, Improving the quality of citizen contributed geodata through their historical contributions: The case of the road network in openstreetmap, ISPRS Int. J. Geo-inf. 7 (7) (2018) 253:1–253:22.

[38] M. Minghini, F. Frassinelli, OpenStreetMap history for intrinsic quality assessment: Is OSM up-to-date? Open Geospatial Data Softw. Stand. 4 (1) (2019) 9:1–9:17.

[39] J.M. Almendros-Jiménez, A. Becerra-Terón, M. Torres, Integrating and querying OpenStreetMap and linked geo open data, Comput. J. 62 (3) (2019) 321–345.

[40] J.M. Almendros-Jiménez, A. Becerra-Terón, M. Torres, The retrieval of social network data for Points-of-Interest in OpenStreetMap, Human-Centric Comput. Inf. Sci. 11 (10) (2021).

[41] J.M. Almendros-Jiménez, A. Becerra-Terón, Analyzing the tagging quality of the Spanish OpenStreetMap, ISPRS Int. J. Geo-Inf. 7 (8) (2018) 323:1–323:26.

[42] R. Bamford, V. Borkar, M. Brantner, P.M. Fischer, D. Florescu, D. Graf, D. Kossmann, T. Kraska, D. Muresan, S. Nasoi, M. Zacharioudakis, Xquery reloaded, Proc. VLDB Endow. 2 (2) (2009) 1342–1353.

[43] J. Robie, D. Chamberlin, M. Dyck, J. Snelson, XQuery 3.0: An XML Query Language, Tech. rep., World Wide Web Consortium (W3C), 2014, https://www.w3.org/TR/xquery-30/.

[44] P. Gómez-Abajo, E. Guerra, J. de Lara, M.G. Merayo, A tool for domain-independent model mutation, Sci. Comp. Prog. 163 (2018) 85–92.

[45] P. Gómez-Abajo, E. Guerra, J. de Lara, M.G. Merayo, Wodel-Test: a model-based framework for language-independent mutation testing, Softw. Syst. Model. (2021) (in press).

[46] M.G. Merayo, R.M. Hierons, M. Núñez, A tool supported methodology to passively test asynchronous systems with multiple users, Inf. Softw. Technol. 104 (2018) 162–178.

[47] M.G. Merayo, R.M. Hierons, M. Núñez, Passive testing with asynchronous communications and timestamps, Distrib. Comput. 31 (5) (2018) 327–342.

[48] R.M. Hierons, M.G. Merayo, M. Núñez, Bounded reordering in the distributed test architecture, IEEE Trans. Reliab. 67 (2) (2018) 522–537.