A Semi-Automated System for Exploring and Fixing OSM Connectivity

Fares Tabet

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

University of Washington

2020

Committee:

Mohamed Ali, Chair

Peiwei Cao

Program Authorized to Offer Degree:

Computer Science and Systems

University of Washington

Abstract

A Semi-Automated System for Exploring and Fixing OSM Connectivity

Fares Tabet

Chair of the Supervisory Committee:
Mohamed Ali
Department of Computer Science and Systems

Routing engines and navigation services are among the top applications that take advantage of the OpenStreetMap (OSM) collaborative project. With that said, the underlying road network data provided by the OSM public geographic datasets must be as accurate as possible for these services to work correctly. This means that road networks must be fully connected, and other constraints such as turn restrictions, road directionality and correct road classification must be respected. However, being an open-license project with around 7 million users and a daily average of about 3 and half million map changes, errors in the data are far from lacking. Issues like misclassified road segments, incorrect connections and gaps in road networks are fairly common, and they pose a complex yet notable obstacle that jeopardizes the accuracy and reliability of routing services that rely on OSM data. In this research, we demonstrate and analyze a system we have developed (the OSM connectivity system, or OCS) that tackles and remedies all sorts of connectivity errors in an OSM road network graph. The system enables the end-user (cartographer) to discover errors and achieve full connectivity in any chosen OSM road network graph. It

automatically detects connectivity errors that otherwise require an extensive manual process to discover, accepts user input through its friendly graphical user interface, and allows for further error investigation, fix suggestion, and easy access to official editing tools.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

OpenStreetMap (OSM) is a rapidly growing open license project that offers map services and geospatial data sets which are made available by editors and cartographers all over the world. It currently serves over 7 million users [1] in various applications and services such as routing engines, global positioning systems, and other G.I.S. systems.

Every day there are about 4.5 million map changes. With such huge number of daily contributions, errors are bound to happen, and in fact often do. Incorrect connections, gaps, road misclassifications and broken links are common connectivity errors found in the road network data of a given region. These errors translate into scenarios like highways that exit directly onto residential roads, U-turns with wrong directionality, and roads that are isolated from other road segments in the road network graph. These faults pose serious obstacles to the reliability of any routine engine or entity relying on OSM road network data.

In addition to the connectivity errors mentioned above, the road network graph of a given region must be fully connected in order for routing engines to properly function. For example, if a user wants to go from their home (point A) to their relative's home (point B), the routing engine will ideally route them through the highest classified road network for as long as possible since these roads are considered to be the widest and fastest to drive on (freeways). Such a trip will typically consist of a few low-class road segments (like residential roads) that reach a highway access point from point A, then the bulk of the trip would be spent on freeways as they are the

fastest option before finally driving again through lower-class roads that exit the freeway and reaching point B. However, if connectivity errors exist in the freeway road network, the routing engine will incorrectly route the user to go through more lower-class roads and take more detours, decreasing the time spent on freeways and increasing the total travel time.

On top of accurate routing, fixing connectivity errors also has various significant benefits. It reduces travel time and therefore saves money and resources by spending less time on the road and using less gas. It increases driving options as more routing options are made available when there are more options to go from point A to point B, which in turn distributes traffic density more efficiently. A fully connected road network also improves reachability and can even in- creases the number of transportation modes (when lower-class road networks are fully connected, biking is made possible). The benefits of an optimal road infrastructure on a region's overall socio-economic productivity have long been studied, and our system aims at achieving them by providing adaptable solutions for detecting errors in road network data and suggesting connectivity fixes for them.

Our research offers reasonable contributions in the areas of OSM, road networks and graph theory, specifically surrounding the problem of optimally connecting disjoint directed subgraphs. Automated error detection and fix suggestions for OSM road network data are made available in an intuitive never seen before manner. In fact, no other tool as far as we know tackles OSM road networks and offers complex solutions for connectivity errors. Algorithmically, the problem of suggesting connectivity fixes to real disconnected subgraphs is truly a challenging one. When dealing with real life roads, there is a plethora of constraints to be considered such as directionality, turn restrictions, shortest distance needed to connect them, fault tolerance, minimal number of

roads needed to connect them, etc. We are dealing with thousands of differently classified road segments within different road network graphs. Additionally, there are multiple combinations to the problem and different orders of connectivity: the more the subgraphs the more there are different ways to connect them. There is also the reality of cascading consequences: if a road segment is re-classified to fix a road network's connectivity, it might break the connectivity of already existing road networks as we are essentially dealing with a single multi-layered composite road network graph.

Existing research surrounding strongly connected components and OSM road networks and quality assurance have aided our research as is expanded in Chapter II. However, no related work directly scratches the issue of road network connectivity and how to optimally connect them. Before we dive into the innards of OCS, we define core definitions and nomenclature in Chapter III. We then describe the algorithms and methods used to develop our system in Chapter IV followed by the architecture used to implement the said methods in Chapter V. We demonstrate the capabilities and showcase results of our system for different countries in Chapter VI.

II. RELATED WORK

A. Strongly Connected Component (SCC)

The proposed system and its contributions fall under different buckets, the first of which is the detection of strongly connected components (SCC). By definition, a strongly connected graph is a graph in which every vertex is reachable from every other vertex. SCC detection nowadays has many uses in various field such as analyzing web graphs [2], social media networks [3] [4], model checking for state space graphs [5], reinforcement learning [6], mesh refinement [7], computer aided design [8] and scientific computing [9].

A considerable amount of research has been (and is still being) done on SCC detection, and we can classify these researches into two different kinds: parallel SCC detection algorithms and sequential SCC detection algorithm. Larger datasets and dynamic graphs require parallel detection algorithms (the use of multiple CPU cores or GPUs simultaneously) as the complexity of the task becomes more challenging when the dataset grows. Such algorithms include the MG-Hybrid [10], the Multistep method [11], the Forward-Backward (FW-BW) method [12] and Orzan's coloring method [13]. The second kind of algorithms are the sequential algorithms. They are a better fit for smaller static datasets which are abundant when it comes to individual road networks. Notable sequential SCC algorithms include Kosaraju's [14], Dijkstra's [15], and Tarjan's [16], which OCS heavily draws upon. We are essentially detecting subgraphs and trying to connect them into an SCC as best we can. As detailed in the following chapters, OCS makes strong use of graph theory

and algorithms such as Tarjan's SCC detection algorithm in its design of an algorithm for suggesting connectivity fixes.

B. OpenStreetMap (OSM) and road networks

First and foremost, this research expands on the published demo paper showcasing this system we built [17]. As a result of its continuous exponential growth, OSM road and transportation network data is being used more and more in applications. These applications include but are not limited to autonomous car space detection (22), road elevation [18] and inclination [19] [20] estimation, transportation network partitioning [21], inter- vehicle communication simulation [22] and graph reduction and interpolation [23].

Even though the advantages of GIS systems [24] have always been clear, correctly charted digital road network and infrastructure data is of utmost importance now more than ever as our world relies heavily on digital datasets. As mentioned earlier, OSM is a huge open-source project with millions of users [1], which means that errors and faults in the data are very likely to occur. It is no surprise that wrong data will lead to wrong results, and there is no exception when it comes to road network data. Proof of errors in the road network data directly reflected onto routing engine results can be found in researches and systems that tackle routing error discrepancies when different underlying map providers are involved [25]. OCS offers solutions for detecting and fixing different errors in OSM road network data, making the overall datasets more reliable and accurate for all of its userbase.

C. Quality Assurance

Various OSM quality assurance tools already exist such as Osmose [26], JOSM/Validator [27] and OSM Inspector [28]. Additionally, research has also been conducted on mechanisms to further improve the OSM data quality [29], some even targeting road networks [30]. However, most of the available tools do not provide suggestions for fixes, and none of them tackle the crucial problem of achieving road network connectivity. OCS is also adaptable as it can cover any region in the world, and it has a user-friendly graphical user interface which makes rectifying road network connectivity errors even more intuitive.

III. PRELIMINARIES

Road segments, also referred to as “ways”, are classified into seven main geographical entities. In the descending order of importance, a road segment is either a: motorway, trunk, primary, secondary, tertiary, residential, or unclassified road. Motorways (also known as freeways) are restricted access divided highways, usually with at least two running lanes and shoulders. Trunks are the most important roads in a country’s system, which are not motorways (they are not necessarily a divided highway). After that come primary, secondary and tertiary roads which are based on length, width and significance. Residential roads are road segments that offer access to housing areas without the purpose of connecting settlements together. Unclassified roads are minor roads that serve purposes other than connecting properties, they are the least important roads in a country’s system [31].

A road network graph G is a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(u, v) \mid u, v \in V\}$. That is, each edge can be followed from one vertex to another vertex. In terms of OSM data terminology, OSM ”nodes” represent vertices and OSM ”ways”(or road segments) represent edges.

With that said, every road classification described above has its own road network graph. For instance, the motorway road network graph is the graph exclusively made out of motorway road segments, $G_{Motorways} = (V, E)$ where $E \subseteq M$, and M represents the set of all motorway road segments.

Although these road networks can exist separately, they are often grouped together to achieve a connected graph. Just like in real life, when you drive from your workplace to your residential street, you are following a path of interconnected road segments which are a subset of the seven different road class we mentioned earlier. Higher-class road segments are connected to each other via lower class road segments, i.e. motorways are often connected by trunks, and trunks are often connected by primaries (and so on going down the road hierarchy). Accordingly, we attribute "levels" to road networks which contain a grouping of different road classes. The highest road network level (also referred to as level 1) is the road network graph made out of the highest road classes in a country, usually motorways and trunks. For example, the highest road classes in New Zealand are motorways and trunks, so G_{level1} in New Zealand = (V, E) where E ⊆ {M, T}, and M and T represent the set of all motorway and trunk road segments respectively. In the Fiji Islands' case, there are no motorways or trunks, so G_{level1} in the Fiji Islands = (V, E) where E ⊆ {P, S}, and P and S represent the set of all primary and secondary road segments respectively.

IV. METHODOLOGY

We can divide the overall OCS workflow and process into 4 main steps: extracting and preparing the data, processing it into custom data structures, detecting road network errors, and suggesting connectivity fixes.

A. Data preparation

The OSM geographical map data is generally made available in protocol buffer binary format (.pbf) or .osm format, which are extended versions of XML. This crowdsourced data is made available under the Open Database License; however, it cannot be used as-is and needs to be converted into other formats to be used by our algorithms. Moreover, this raw data contains granular levels of details for the selected region such as information about amenities like restaurants. As mentioned in Section III, we are mainly interested in two geographical entities - ways and nodes - and their dependent objects. Apart from these we are also interested in extracting relations defined for ways and nodes. OSM 'relations' define geographic relationships between ways, nodes and other relations. For example, a relation R can define that "no U-Turn" can be made at a node N when going from a way W1 to way W2.

We perform a first level of optimization at this stage by extracting only the required data and the tags (i.e. nodes, ways, relations with required metadata/tags), including their dependent objects such as the nodes that make up ways, ways of relations, relations of other relations. We convert

the raw data to .xml files by using OSMConvert [32] and filter it by using OSMFilter [33], two powerful command line tools. Once we have the data filtered and in XML format, it can readily be used by algorithms and programs for further processing.

B. Pre-processing data

We designed our system to be object-oriented, thus all operations performed mainly include three objects - Node, Way and JunctionNode. We also define a Country object which stores all the metadata and the road-hierarchy for a particular country. Since XML format does not provide the fastest lookup for large data sets like ours (even after filtering), we transform this data to a more suitable format. We parse this data through our system and save it as a collection of maps. Having the data as key-value pairs provides both fast and easy access to any entity we need. This collection is distributed into 4 major maps: Nodes, Ways, Relations and Junction nodes. All 4 hash maps follow a uniform structure of key being the entity id (identifier) and its corresponding value being the information about that entity. The Node map contains information such as each node's id, list of ways, and road classes that it is a part of. The way map is comprised of information such as each way's id, road class, geometry or nodes present in the way, start and end nodes, and its directionality (i.e. whether the node is bidirectional or unidirectional). Similarly, the Relations map contains the relation's id, nodes, and ways. The Junction nodes map houses nodes which are either the start or end node of a way (i.e. junction points/ vertices in a graph). This map contains the basic geometry of the entire road network graph.

Once OCS processes the pre-filtered OSM files, it creates the above collection of maps and saves them as JSON files to avoid pre-processing again in case the data remains the same.

While the data is being parsed, we simultaneously address the following issues:

- **Detecting road hierarchies:** We enumerate the road-classes available in OSM data as per their importance. The road- hierarchy for each country is initialized to the lowest road level. Once we start reading the XML files into objects, the road-type tag of each way object is compared to the existing road-hierarchy of the country and updated if it is at a higher level than the existing road-hierarchy.

- **Resolving the 180-degree issue on flat maps:** The 180th meridian or antimeridian is the longitude which is both 180-degree east and west of the prime meridian (the zero-based reference on which all longitudes are placed). In most software or geographic data, earth is represented as a rectangle, and the antimeridian acts as the edge of the map. However, OSM data maintains two copies of antimeridians to represent traversal directions: +180 degree and -180 degree. For each geographic node that falls on these antimeridian lines, OSM represents the same node with two copies. For example, a single node that falls on an antimeridian can be represented by two different OSM nodes with coordinates like (-16.7935583, 180.0000000) and (-16.7935583, -180.0000000). Even though these two OSM nodes represent the exact same location, the data has no way object connecting these two nodes. This causes the generation of disconnected components at these nodes even though we should achieve continuous connectivity. To resolve this issue, we resorted to generating hypothetical ways that connect these two nodes, which we refer to as complementary nodes. The idea behind this is to collect all the nodes that lie on the antimeridian for a particular country, identify the pair of nodes with same latitude but complementary longitudes, and create a hypothetical way which has starting point as one

node and ending point as the other node in the pair (considering it to be bi-directional). This enables us to form a continuous connection passing through these two nodes, rectifying any invalid disconnections caused by the antimeridian. This is illustrated in Figure 1 where N1 and N2 lie on antimeridian and have same latitudes but complementary longitudes (+180 and -180), even though they represent the same geographic location. Since N1 and N2 do not connect through any ways, we cannot traverse from W1 to W2 or vice-versa depending on the directionality even though these ways are actually connected. Adding the hypothetical way AM_W between N1 and N2 resolves this issue.

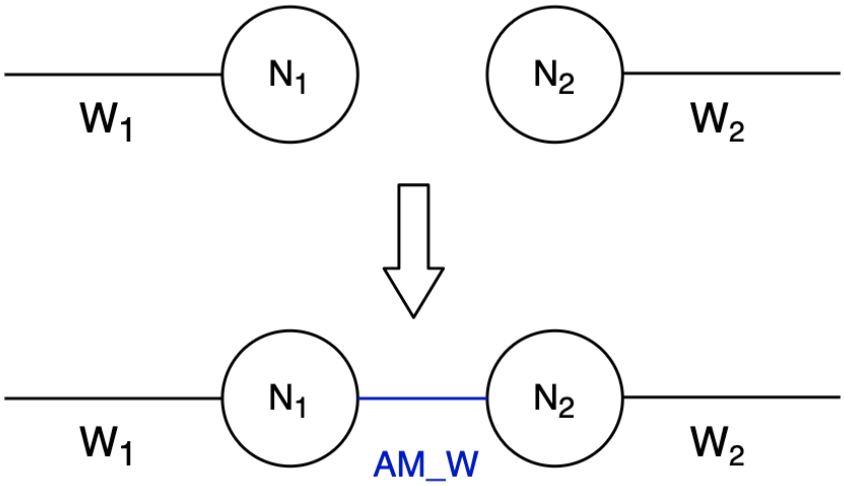

Figure 1. Adding hypothetical ways

- **Detecting ferry terminals:** Countries such as New Zealand (which are a collection of islands) are interconnected by waterways. To achieve connectivity for a particular level such as level 1 (connect motorways with least number of trunks), we need to connect all motorways present in all the islands. We propose to achieve this by firstly connecting all the motorways in the individual islands using trunks, connecting the resultant graphs on

each island to the nearest ferry terminals, and finally connecting these ferry terminals together via waterways. The prerequisites to do this is to identify the waterways and the ferry terminals. We perform this identification while parsing the XML files by adding a ”ferry” attribute to the way object if it is a waterway and by collecting the nodes with ”amenity” tag as ”ferry terminal”.

- **Detecting and resolving T-intersections:** Each way is composed by a collection of nodes: a start node, an end node and the intermediate nodes in between them. For our algorithms we focus only on the starting and ending nodes of the ways, which we refer to as junction nodes, since these are sufficient in and of themselves to achieve connectivity (solely connecting the start and end nodes of ways together is practically how we connect them). However, there are a few scenarios as illustrated in Figure 2 where way $W1$ has N_S as the starting node, N_E as the end node and $N2$ as an intermediate node; and way $W2$ has $N1$ as the starting node and $N2$ as the end node. Now since our algorithms are based on start and end nodes, they will not be able to make out a path from $W2$ because $N2$ would be considered as a dead end (it isn’t a junction between another start/end node). To enable such traversals, we divide the way at points where an intermediate node is either the start or end point of other ways into sub-ways we create. As you can see in the illustration, we divided $W1$ into 2 different ways: $W1A$ with N_S as the start node and $N2$ as the end node, and to $W1B$ with $N2$ as the start node and N_E as the end node, enabling traversal between N_S, $N2$, $N1$ and N_E. We detect all such T-shaped intersections and process them to enable all traversal paths.

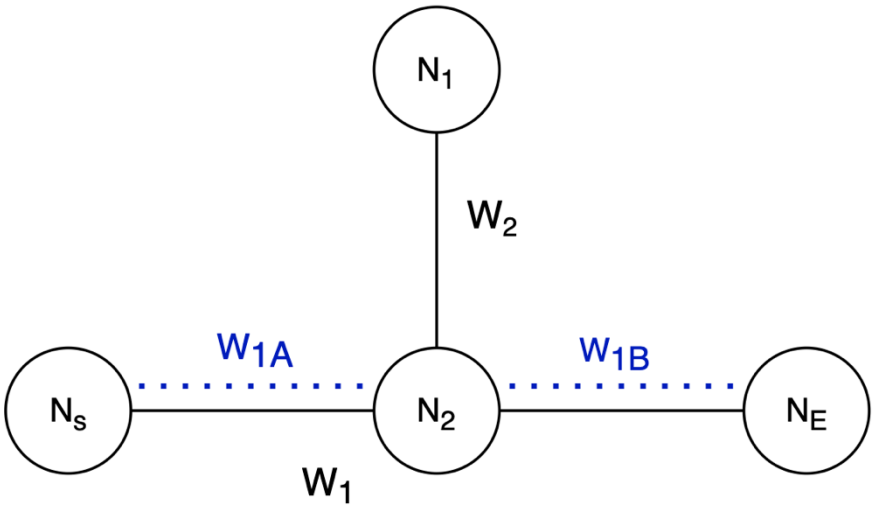

Figure 2. Detecting T intersections

- **Turn restrictions:** It is crucial to consider traffic signs while applying traversal algorithms. In our work, we focus mainly on four types of restrictions: no U-turn, no right turn, no left turn, and no straight on. In OSM, such restrictions are expressed as relations. Each relation has ’from’, ’via’ and ’to’ tags. These tags imply that for a particular relation like a “no U-turn”, one is not allowed to traverse from the way objects tagged as ’from’ to way objects tagged as ’to’ through a node tagged as ’via’. For our algorithm, we find the starting nodes for the ’from’ ways and tag these nodes as ’from’. Similarly, we tag ’to’ nodes and ’via’ nodes. If a tag combination of the form ’from-via-to’ is encountered while traversing, then the node tagged as ’to’ is not considered for further traversal and not included in the resulting graph. In other words, for a particular relation R: if the currently explored node is tagged as ’via’ for R, its parent is tagged as ’from’ for R, and its child is tagged as ’to’ for R, we do not consider its child node for further traversal. Figure 3 illustrates this by showing a restriction R in which x is tagged as ’from’, N1 as ’via’, and N2 is tagged as

'to': it dictates that one cannot traverse from W1 to W2 via N1 (as represented by the red dotted line).

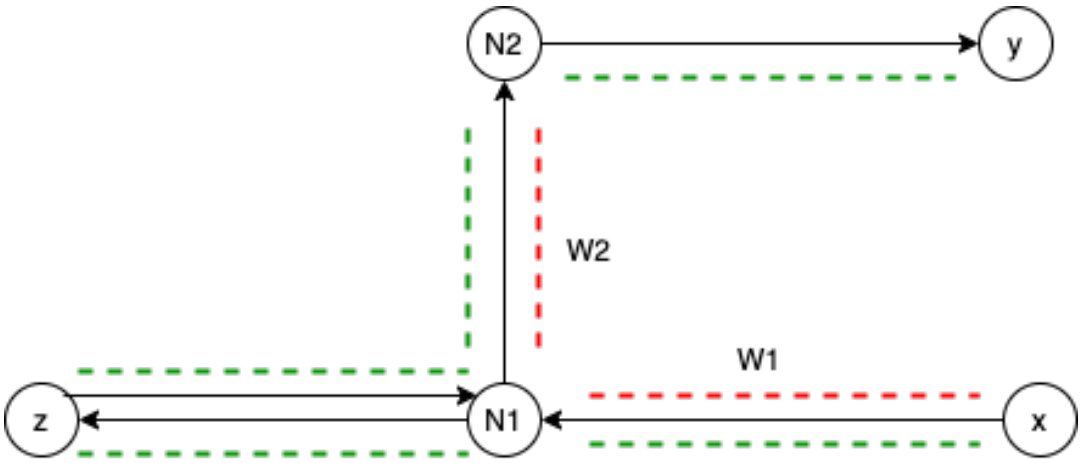

Figure 3. Incorporating turn restrictions

Let us assume x has just been traversed by our algorithms and N1, the child of x, is the current node being explored. Even though N1 has two children (z and N2), only z is added in the child queue of N1 for further exploration. N2 is not added as a child of N1 because it would complete the combination 'from-via-to' R since its parent is the 'from' node x, N1 itself is the 'via' node and N2 is the 'to' node. With this approach, we are able to effectively implement turn restrictions.

C. Detecting errors in OSM road network

There are some restrictions imposed by the OSM documentation on the possibility of having two different road classes connected to each other [34]. These restrictions dictate which road classes can be directly connected to each other. For example, it is forbidden for a motorway road segment

to be directly connected to a residential road segment, but it is the norm for it to connect directly to another motorway or a trunk link. To detect and extract such errors, the system cycles through all the different road segments and imposes condition checks that enforce the connectivity restrictions between different road classes. Once a violation is found, the system records the specific error, adds it to an "incorrect connection" record file to be downloaded by the user, and displays it to on its graphical user interface. For each incorrect connection found, the user is given information about the exact incompatible road segments and classes that cause the issue, as well as the exact junction node that connects them. OCS also gives the user direct access to links that will open up the error on different official OSM editors for a quick and convenient way of directly fixing the issue as is shown in Figure 4.

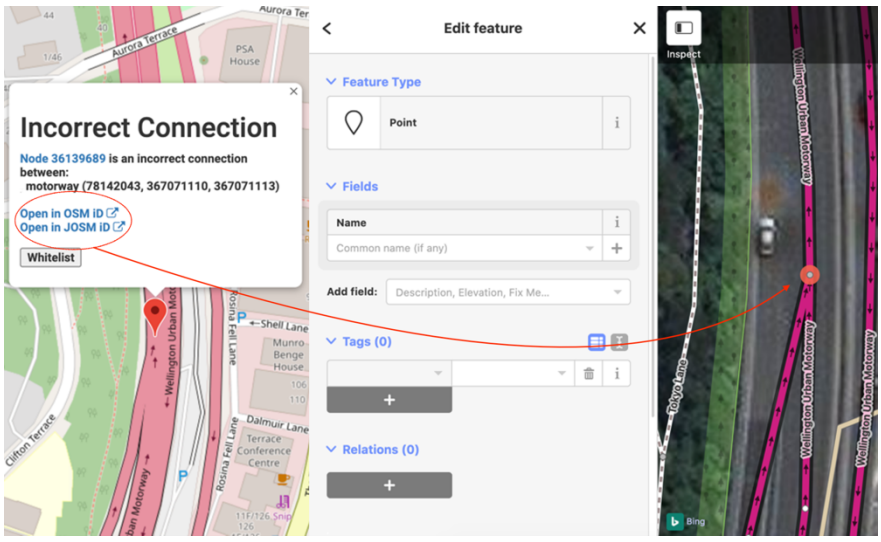

Figure 4. An incorrect connection

D. Suggesting connectivity fixes

The true innovation our system offers lies in its ability to suggest fixes to achieve full connectivity for any desired road network. A road network graph is fully connected if every node in that graph is reachable from every other node (a strongly connected graph).

Algorithm 1: Achieving full connectivity for disconnected subgraphs

Input : List of Higher-Level Road Network Classes $HRNC$ in which we would like to achieve connectivity, list of Lower-Level Road Network Classes $LRNC$, used to connect the $HRNC$

Output: Fully Connected Graph G_{FC}

Find list of Disconnected Road Network Graphs $DRNG$ using a graph traversal algorithm from G_{HRNC}. $DRNG$ = $\{G_{disconnected_1}$, $G_{disconnected_2}$,$G_{disconnected_3}$, ... , $G_{disconnected_n}\}$

$\forall G_{disconnected} \in DRNG$ extract the outgoing and incoming nodes i.e N_i and N_o

Pick at random $G_{disconnected} \in DRNG$. Set $G_{source} \leftarrow G_{disconnected}$

Remove G_{source} from $DRNG$

while *$DRNG$ is not empty* **do**

  $\forall N_o \in G_{source}$ as source nodes, run a graph traversal algorithm until it encounters a Node N ($N \in HRNC$ and $N \notin G_{source}$)

  if *we encounter N which matches the above condition. Say $N \in G_B$ (closest graph to source)* **then**

    Find to-and-from paths via allowed $LRNC$ i.e $P_{source \leftarrow B}$ and $P_{B \leftarrow source}$

    Pick the paths with the least number of nodes

    $G_{source} \leftarrow G_{source} \cup G_B \cup P_{B \leftarrow source} \cup P_{source \leftarrow B}$

    Keep updating $G_{source} \leftarrow G_{source} \cup G_B \cup P_{B \leftarrow source} \cup P_{source \leftarrow B}$ with paths from $P_{source \leftarrow B}$ and $P_{B \leftarrow source}$ till G_{source} is a strongly connected component or $P_{source \leftarrow B}$ and $P_{B \leftarrow source}$ are empty.

    Remove G_B from $DRNG$

  end

  if *graph traversal does not encounter a target Node N* **then**

    $G_{source} \leftarrow G_{source} \cup G_B$

    Remove G_B from $DRNG$

    Pick at random $G_{disconnected} \in DRNC$. $G_{source} \leftarrow G_{source} \cup G_{disconnected}$

  end

end

return G_{source}

Algorithm 1 portrays how our system finds disconnected portions in a graph and how it computes their respective connectivity fixes. First, the system detects disconnections in the given list of road network graph. For example, if we would like the system to suggest connectivity fixes in a motorway road network of a country and use trunks to suggest fixes, the system would detect disconnections in the motorway road network graph. Finding disconnections is done using graph traversal algorithms, in our case we have used Breath First Search (BFS) to do so. Applying a graph traversal algorithm in the same example above, we find a list of disconnected subgraphs of motorways we name DRNG. From this list we pick at random a disconnected subgraph which will be used as a source graph *Gsource*. *Gsource* is then used as the starting point to connect all the disjoint subgraphs. Since road network graphs are considered to be directed graphs, every disconnected subgraph will have incoming edges and outgoing edges (roads that enter and exit the subgraph). For each of these subgraphs we create a list of vertices of endpoints of inward edges and another list of vertices which endpoints of outward edges. Incoming vertices are used as incoming nodes for a particular subgraph whereas outgoing vertices are used as nodes used to traverse away from a subgraph. Using the same graph traversal algorithms, we traverse away from *Gsource* using the outgoing vertices as starting point and try to find any incoming vertices from another disjoint subgraph. Referring back to the same example, we traverse using the road network class roads that are given by the user (in this case, using trunk roads). Once we encounter an incoming node from another disjoint subgraph, we connect the two subgraphs using the paths we computed using the graph traversal algorithm. In our case since we used BFS algorithm, this connects the two subgraphs using least number of hops/nodes between the two subgraphs. We know that there will exist multiple paths between any two graphs, so we first try to connect the two using the shortest path from source graph to the newly found subgraph (say Graph A) and vice

versa. After this step, we have a newly formed graph which contains source graph *Gsource*, Graph A and to-and-fro paths from the two. From there, we check whether this graph is a strongly connected component using Targan's Strongly Connected Component Algorithm. If this new graph is not a Strongly Connected Component (SCC), we continue appending the to-and-fro paths into this newly formed graph in order of least number of nodes in the paths first till this graph is a SCC or till we have exhausted all the paths between them. This resultant graph becomes a new *Gsource*. Outgoing vertices of this updated *Gsource* are used to find incoming vertices from the remaining disjoint subgraphs.

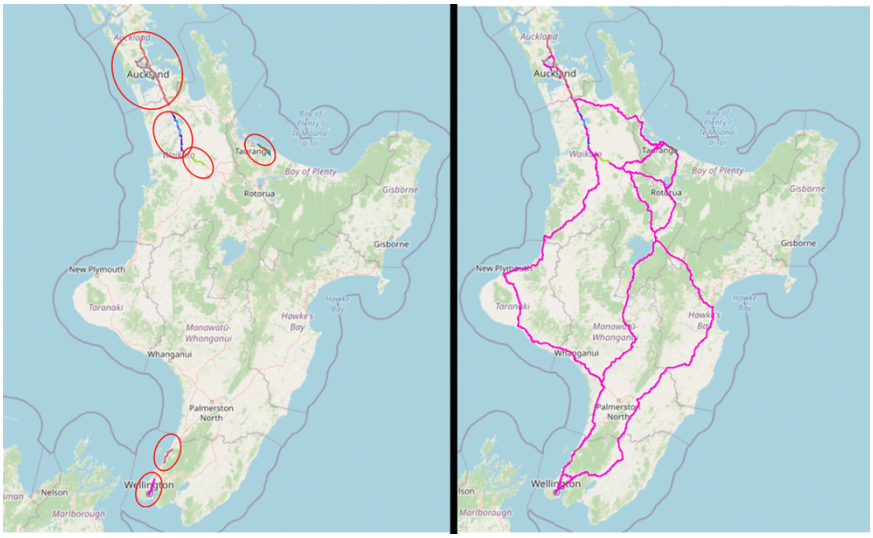

Figure 5. Connecting subgraphs in New Zealand's level 1 road network

Sometimes *Gsource* cannot find a disjoint subgraph despite other subgraphs are present, these cases generally arise in scenarios when road network graphs are spread across multiple islands or due to road constructions. When faced with this issue, we randomly pick a disjoint subgraph and append it to *Gsource*. This gives the Source graph new outgoing vertices and help spread across a

new region of graph. The algorithm continues to run the same process described above till all the disconnected subgraphs are appended to G_{source}. This newly updated G_{source} contains all the paths which are required to connect the disjoint subgraphs and hence we have achieved a connectivity fix for the graph.

Figure 5 above visualizes the before and after of running algorithm 1 on New Zealand's level 1 road network (Northern island). On the left, the different subgraphs detected are shown, and on the right the final connectivity fix result is displayed.

V. IMPLEMENTATION

Our system contains 2 major components: the backend module which runs all the algorithms and contains the business logic, and the frontend module which generates visualizations.

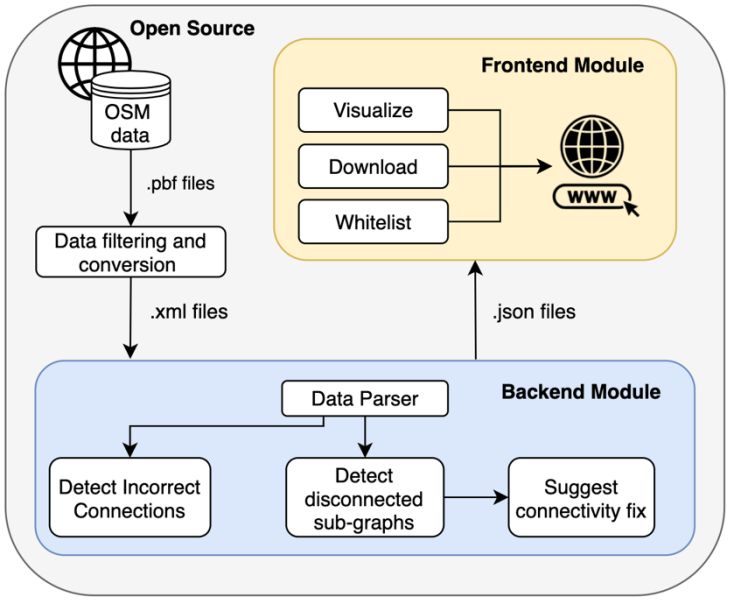

Figure 6. OCS System Architecture

A. Backend

The backend is an independent module that takes already prepared XML data files as input (as explained in Section IV-A). The data fetched is refreshed daily to keep up with daily map changes. From there, the logic and techniques mentioned in Section IV-B are applied to parse the XML files into data structures and make them ready as input for our algorithms. Once our algorithms run,

JSON files representing the results of all operations are generated and fed to the frontend. The backend module was built using the general-purpose C# programming language.

B. Frontend

Built using the Microsoft ASP.NET MVC framework, the fronted module acts the graphical user interface from which the user can interact with OCS. It allows the user to:

- Visualize different road networks, waterways, ferry terminals, incorrect connections, disconnected components, and connectivity fixes with different colors to denote their isolation.
- Pop up red markers for all incorrect connections. When clicked, the markers provide in depth details about the error detected and allow one-click access to major OSM editing websites in which the error would already be loaded in.
- Flag (or 'whitelist') an error as a false positive. This reversible action will prevent said error from appearing when other users are detecting errors.
- Download all the detected errors and connectivity fixes as JSON files for further analysis.
- Hover on an entity to get more information such as the number of ways of each type used to achieve connectivity, the id of a road segment, a link to open it on the OSM website etc.

VI. EXPERIMENTS

In this section, we evaluate the performance of OCS. First, we describe the environment of the experiments. Then, we explore and elaborate the results in terms of incorrect connections and connectivity fixes across different countries.

A. Experimental Setup

For all our experiments, we used the most up-to-date data for New Zealand (NZ), Venezuela (VZ), Serbia (SB) and Fiji (FJ) that was retrieved on December 10th, 2020 from the OSM servers [35]. All evaluations were conducted on a Macbook (macOS 11.0.1) with a 2.6 GHz Quad-Core Intel Core i7 processor and 16GiB RAM.

B. Incorrect Connections

We ran our algorithms to detect incorrect connections across the four most significant road classes: motorways, trunks, primaries and secondaries (note that Fiji does not have any motorways or trunks). Table I shows the exact numbers our algorithms go through for each country.

Country	Ways	Nodes	Junction Nodes
New Zealand	43898	368008	37540
Fiji Islands	1126	18972	1001
Venezuela	32755	377710	25852
Serbia	28586	579198	25579

TABLE I Experimental data for each country

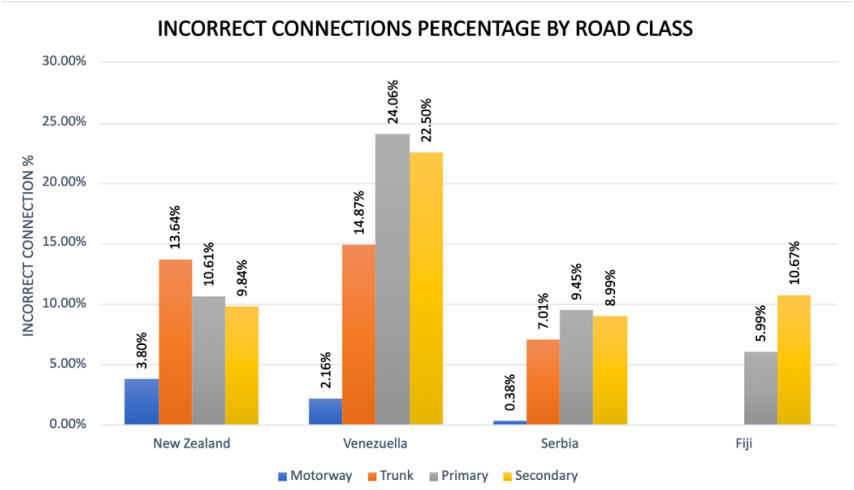

Figure 7. Incorrect connections statistics

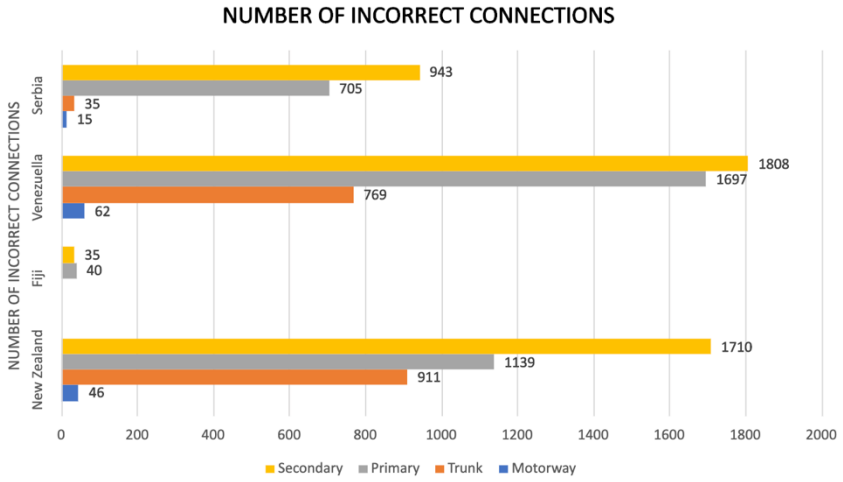

Figure 8. Incorrect connections statistics

Time wise, detecting incorrect connections for all our experiments was an almost instantaneous process due to the efficiency of our algorithms and data structures. As can be seen in Figure 8, OCS was able to detect a significant amount of incorrect connections across these different classes. In most cases, it seems that the lower the road class the higher the number of incorrect connections found. This makes sense because in general, the lower the road class the larger its road network in terms of the amount of roads it is comprised of (i.e. there are less motorways than trunks, and less trunks than primaries). To get more concrete results and eliminate this bias, we have extracted the percentage of incorrect connections by road class (i.e. number of motorway incorrect connections/total number of motorway connections) and have displayed the results in Figure 7. The bar chart illustrates a very powerful and concerning result with some error percentages close to 25% (like VZ's primaries). To put that into perspective, this means that approximately 1 out of 4 primary road junctions in Venezuela go against the connectivity rules outlined in the OSM documentation [34]. Another observation made is that the error percentage for motorways is relatively low compared to other road classes. This might be because there are much less motorways than other road classes which makes manual error detection more manageable and effective, or because of greater meticulousness by cartographers due to the sheer importance of these roads. As we move away from motorways, there is much more data to go through which makes it exponentially more difficult to detect errors manually, resulting in higher error percentage.

C. Connectivity Fixes

We demonstrate our proposed algorithm for suggesting fixes to achieve full connectivity for the two major islands in New Zealand (Northern and Southern), three major islands in the Fiji Islands, and Venezuela. Table I shows the total number of nodes, ways, and junction nodes that our algorithms go through for each country (As discussed in Section IV-B).

To achieve connectivity for a certain level, our algorithms first detect the different disconnected subgraphs within that level, then proceed by connecting them. For each country and level, Table II illustrates the number of subgraphs detected, number of ways to be added (minimum number of road segments needed to connect them), total ways (total number of road segments that make up a connectivity fix) and the time it took to generate the results. To get a better insight, we have recorded iteration of different connectivity fixes and plotted them. Figure 9 illustrates how many total ways it takes to connect the subgraphs until all of them are connected. Typically, connectivity the subgraphs each step of the algorithm our algorithm starts connecting all the subgraphs by connecting every pair of subgraphs into one. This pairwise connection is done by adding ways and it can be observed that as we connect and merge more subgraphs, the total number of ways to be added to achieve full connectivity keeps increasing.

Country	Connectivity Level	subgraphs *(S)*	Ways needed *(WA)*	Total ways *(WC)*	Time (h:mm) *(T)*
New Zealand	Level 1 (Northern Island)	10	3087	3696	1:22
New Zealand	Level 2 (Northern Island)	2	328	5385	0:12
New Zealand	Level 1 (Southern Island)	3	1193	1294	0:04
New Zealand	Level 2 (Southern Island)	2	982	3898	0:16
Fiji Islands	Level 1 (3 main Islands)	2	7	742	0:01
Fiji Islands	Level 2 (3 main Islands)	3	24	1077	0:02
Fiji Islands	Level 3 (3 main Islands)	4	3	1080	0:04
Venezuela	Level 1	16	3432	5230	3:41

TABLE II Connectivity information by level and country

As can be seen in Table II, the lower in connectivity levels for the same region we go, the higher the number of total ways that make up the connectivity fix (WC). This reflects the fact that the more road types we want to connect, the more road segments there are to consider (level 2 contains level 1 roads classes, and level 3 contains both levels 1 and 2 road classes).

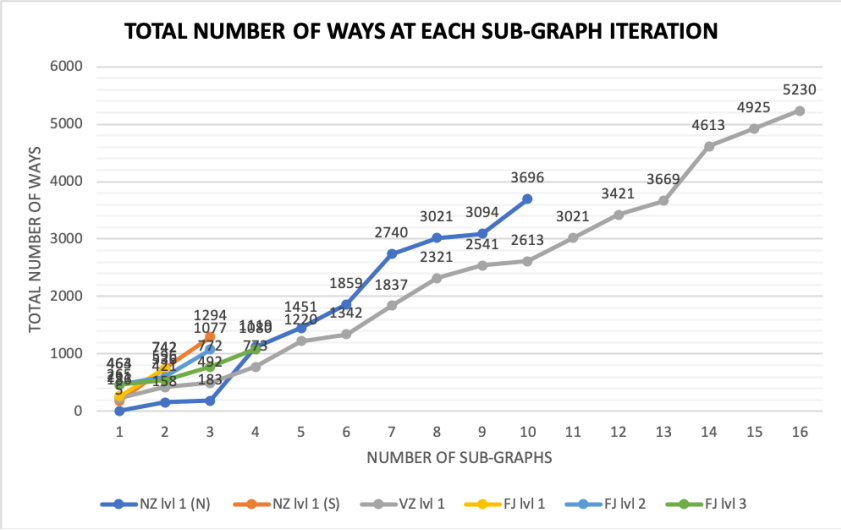

Figure 9. Total number of ways at each connectivity fix subgraph iteration

Another observation tightly linked to the above statement and clearly shown by the trend of every curve in Figure 9 is that an increase in the number of subgraphs leads to an increase in the number of ways required to connect them. For example, consider achieving level 1 connectivity in the Northern island of New Zealand. To connect all motorways using trunks, our system detected 10 isolated motorway subgraphs. It is clear that each iteration of the algorithm when connecting subgraphs requires more ways, as detailed in Figure 9). Once completed, Table II shows us that an additional 3087 ways (trunks) that are not part of these 10 subgraphs were needed to result in a

fully connected graph (which totals 3696 ways composed of motorways and trunks). As we go down to level 2, our aim becomes to connect all level 1 and level 2 roads (motorways and trunks). In this case, only 2 subgraphs are detected. Since these subgraphs are made out of both motorways and trunks, we can safely assume that level 2 connectivity will have more ways than level 1 connectivity because we now also want to connect all trunks on top of all motorways (instead of all motorways using the least amount of trunks as in level 1). Another not so obvious observation to make is that we cannot correlate processing time with the number of subgraphs. As proved in Table II, for the Fiji Islands it takes double the time to connect 4 subgraph (level 3) than it does to connect 3 (level 2). However, it takes more time to connect these 3 subgraphs than it does for the 2 level 1 subgraphs. For New Zealand, it almost takes an extra hour to connect 10 subgraphs (level 1 Northern Island) than it does for 2 subgraphs (level 2 Northern Islands). Contrarily, it takes less time to connect 3 subgraphs (level 1 Southern Island) than it does for 2 (level 2 Southern Island). This is due to the unpredictable circumstances for each subgraph such as its distance from other subgraphs, the number of distinct paths that lead to them, the number of turn restrictions, T-intersection and other complex pathways that lie in between them, etc.

D. Experimental Summary

Our experiments prove that our system is capable of detecting connectivity errors and suggest connectivity fixes across different levels to achieve full connectivity. Across the four major road classes, OCS detected an incorrect connection average of about 10.57% for junction node in New Zealand, 7.53% in the Fiji Islands, 18.75% in Venezuela and 7.57% in Serbia. We were able to suggest connectivity fixes for level 1&2 for New Zealand, levels 1,2&3 for the Fiji Islands and

level 1 for Venezuela by determining the minimum number of ways belonging to the immediate lower level needed to do so.

VII. CONCLUSION

In this thesis, we analyze a system we built to tackle connectivity in OSM road networks. Our algorithms allow for an unprecedented automation of tedious and complex tasks for any given road network such as detecting incorrect connections, detecting isolated subgraphs and suggesting connectivity fix whilst offering the user an intuitive graphical user interface and easy access to official fixing tools. The use of our system is sure to save countless hours on maintaining and improving the accuracy of road network data which is paramount for reliable routing and dependable use for the ever-growing user-base of OSM.

REFERENCES

[1] "Map statistics," December, 2020. [Online]. Available: https://www.openstreetmap.org/stats/data stats.html

[2] A.Broder,R.Kumar,F.Maghoul,P.Raghavan,S.Rajagopalan,R.Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," Computer networks, vol. 33, no. 1-6, pp. 309–320, 2000.

[3] S. Aldegheri, J. Barnat, N. Bombieri, F. Busato, and M. C˘es˘ka, "Para- metric multi-step scheme for gpu-accelerated graph decomposition into strongly connected components," in European Conference on Parallel Processing. Springer, 2016, pp. 519–531.

[4] A.Mislove,M.Marcon,K.P.Gummadi,P.Druschel,andB.Bhattachar- jee, "Measurement and analysis of online social networks," in Proceed- ings of the 7th ACM SIGCOMM conference on Internet measurement, 2007, pp. 29–42.

[5] S. Orzan, "On distributed verification and verified distribution," Ph.D. dissertation, PhD thesis, Free University of Amsterdam, 2004.

[6] S. J. Kazemitabar and H. Beigy, "Automatic discovery of subgoals in reinforcement learning using strongly connected components," in International Conference on Neural Information Processing. Springer, 2008, pp. 829–834.

[7] W. Mclendon Iii, B. Hendrickson, S. J. Plimpton, and L. Rauchwerger, "Finding strongly connected components in distributed graphs," Journal of Parallel and Distributed Computing, vol. 65, no. 8, pp. 901–910, 2005.

[8] A. Xie and P. A. Beerel, "Implicit enumeration of strongly connected components and an application to formal verification," IEEE Transac- tions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, no. 10, pp. 1225–1230, 2000.

[9] A. Pothen and C.-J. Fan, "Computing the block triangular form of a sparse matrix," ACM Transactions on Mathematical Software (TOMS), vol. 16, no. 4, pp. 303–324, 1990.

[10] J.Hou,S.Wang,G.Wu,B.Ma,C.Si,andS.Jia,"Mg-hybrid:Astrongly connected components detection algorithm using multiple gpus," in 2020 IEEEInternationalSymposiumonCircuitsandSystems(ISCAS). IEEE, 2020, pp. 1–5.

[11] G. M. Slota, S. Rajamanickam, and K. Madduri, "Bfs and coloring- based parallel algorithms for strongly connected components and related problems," in 2014 IEEE 28th International Parallel and Distributed Processing Symposium. IEEE, 2014, pp. 550–559.

[12] L. K. Fleischer, B. Hendrickson, and A. Pınar, "On identifying strongly connected components in parallel," in International Parallel and Dis- tributed Processing Symposium. Springer, 2000, pp. 505–511.

[13] S. Orzan, "On distributed verification and verified distribution," Ph.D. dissertation, PhD thesis, Free University of Amsterdam, 2004.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. MIT press, 2009.

[15] E. Wybe and Dijkstra, A discipline of programming. prentice-hall Englewood Cliffs, 1976, vol. 613924118.

[16] R. Tarjan, "Depth-first search and linear graph algorithms," SIAM Journal on Computing, vol. 1, no. 2, pp. 146–160, 1972. [Online]. Available: https://doi.org/10.1137/0201010

[17] F.Tabet,B.H.Patel,K.Dincer,H.Govind,P.Cao,A.Song,andM.Ali, "A semi-automated system for exploring and fixing osm connectivity," in Proceedings of the 28th International Conference on Advances in Geographic Information Systems, 2020, pp. 421–424.

[18] C. Boucher, "Fusion of gps, osm and dem data for estimating road network elevation," in 2013 Fifth International Conference on Compu- tational Intelligence, Communication Systems and Networks. IEEE, 2013, pp. 273–278.

[19] C. Boucher and J.-C. Noyer, "Automatic estimation of road inclinations by fusing gps readings with osm and aster gdem2 data," in 2014 International Conference on Connected Vehicles and Expo (ICCVE). IEEE, 2014, pp. 871–876.

[20] "Dual-gps fusion for automatic enhancement of digital osm roadmaps," in 2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL). IEEE, 2012, pp. 1–6.

[21] M. S. Ahmed and M. A. Hoque, "Partitioning of urban transportation networks utilizing real-world traffic parameters for distributed simulation in sumo," in 2016 IEEE Vehicular Networking Conference (VNC). IEEE, 2016, pp. 1–4.

[22] C.Bewermeyer,R.Berndt,S.Schellenberg,R.German,andD.Eckhoff, "Poster: cosmetic-towards reliable osm to sumo network conversion," in 2015 IEEE Vehicular Networking Conference (VNC). IEEE, 2015, pp. 151–152.

[23] A. Ali, Y. Chen, D. Fuller, and S. Al-Eidi, "Road importance using complex-networks, graph reduction & interpolation," in 2020 Inter- national Conference on Computing, Networking and Communications (ICNC). IEEE, 2020, pp. 855–859.

[24] C. L. Brooks, "Automation technology using geographic information system (gis)," 1994.

[25] A. Bandil, V. Girdhar, K. Dincer, H. Govind, P. Cao, A. Song, and M. Ali, "An interactive system to compare, explore and identify discrep- ancies across map providers," in Proceedings of the 28th International Conference on Advances in Geographic Information Systems, 2020, pp. 425–428.

[26] "Osmose documentation wiki," December, 2020. [Online]. Available: https://wiki.openstreetmap.org/wiki/Osmose

[27] "Josm/validator," June, 2020. [Online]. Available: https://wiki.openstreetmap.org/wiki/JOSM/Validator

[28] "Osm inspector," December, 2020. [Online]. Available: https://wiki.openstreetmap.org/wiki/OSM_Inspector

[29] 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." Multidisciplinary Digital Publishing Institute, 2018.

[30] M. Jilani, P. Corcoran, and M. Bertolotto, “Automated highway tag assessment of openstreetmap road networks,” in Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2014, pp. 449–452.

[31] “Map highway wiki,” December, 2020. [Online]. Available: https://wiki.openstreetmap.org/wiki/Key:highway

[32] “Osm convert,” December, 2020. [Online]. Available: https://wiki.openstreetmap.org/wiki/Osmconvert

[33] “Osm filter,” December, 2020. [Online]. Available: https://wiki.openstreetmap.org/wiki/Osmfilter

[34] “Osm connectivity rules,” December, 2020. [Online]. Available: https://wiki.openstreetmap.org/w/images/1/16/This_table_highlights_how_you_connect_different_types_of_highway%2C_based_on_the_type_of_the_highway._.png

[35] “Geofabrik download server,” December, 2020. [Online]. Available: https://download.geofabrik.de