POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

OpenStreetMap Recursive Street Extraction Algorithm

Tutor: Professor Emanuele Della Valle

Student: Hadi Choueiry
ID: 736513

Academic Year: 2009/2010

Table of Contents

Table of Figures

Acknowledgments

I would like dedicate this work to my parents Sami Choueiry and Maria Abdo, and my two brothers Spiro and Walid for being there for me all the way throughout college. Thank you for always supporting me no matter where I was and for offering me career advice whenever I needed it. Without you I wouldn't have made it here.

I would like to thank Professor Emanuele Della Valle for his guidance throughout my work and for always being helpful.

I would like to thank Maria Antonietta for her constant support and help during those two years.

And of course a big thank you to all my friends and the people I met in Como for the good times we had in school and on the lake of Como and for making this experience an unforgettable one.

Abstract

OpenStreetMap is an open source map containing geographical information. Three key elements make up the map data for OpenStreetMap namely nodes, ways and relations. An OpenStreetMap API allows the insertion, retrieval and deletion of information of data in the map.
The developed application extracts all the streets within a specific bounding box expressed in terms of latitude and longitude. In this thesis three possible algorithms to extract the streets are discussed with their advantages and disadvantages. A recursive algorithm is considered the most appropriate given the requirements of the application.
The recursive algorithm exploits the functionalities in OpenStreetMap API to extract all the streets in a specific bounding box. Starting from the initial node streets are extracted recursively until all the streets with at least one node within the bounding box are extracted. Streets are broken up into links and rearranged in a way to allow their reuse by other applications.
The recursive algorithm is explained in detail describing the flow of the execution and the used methods. A concrete example illustrates step by step the execution of the application and the resulting RDF output is shown to demonstrate the results.
The application successfully extracts the required data and outputs it as required. Few suggestions are given in order to improve the application and enhance its functionality.

1. Introduction

OpenStreetMap is an open source map containing geographical information. The information available in OpenStreetMap could be used by anyone for any purpose.

The goal of this work is to use OpenStreetMap to retrieve information about streets in a specific area defined by its longitude and its latitude. More precisely, given an initial starting node and the dimension of the area of interest around it, the application retrieves all the streets and the nodes in the specified area along with other deemed useful information as the street names, nodes' longitudes and latitudes. After the information is retrieved, it will be rearranged in a way to allow its reuse by other applications for navigational or any other purposes.

The application will return as a result an RDF file listing all the information. RDF is a file format that allows information sharing and merging between different applications even if their underlying architectures are different. RDF is meant to be read and interpreted by computers not by human beings; having said that, the goal of this application is to produce results that could be utilized by other computer applications for different purposes. One possible use could be to use of the results for navigation or for querying geographical data in a specific area and so on and so forth.

In this project, two new concepts are defined junction nodes and links. Below is an explanation of these terms in the context of this application.

Junction nodes and regular nodes

In OpenStreetMap a street is a relation that ties many nodes. A Street must be made of at least two nodes. In other words a street is a collection or an interconnection of many nodes. Nodes could belong to one or more streets.
In this application the terms junction nodes and regular nodes are used to differentiate nodes that belong to more than one street (intersection nodes) from nodes that belong to only one street.

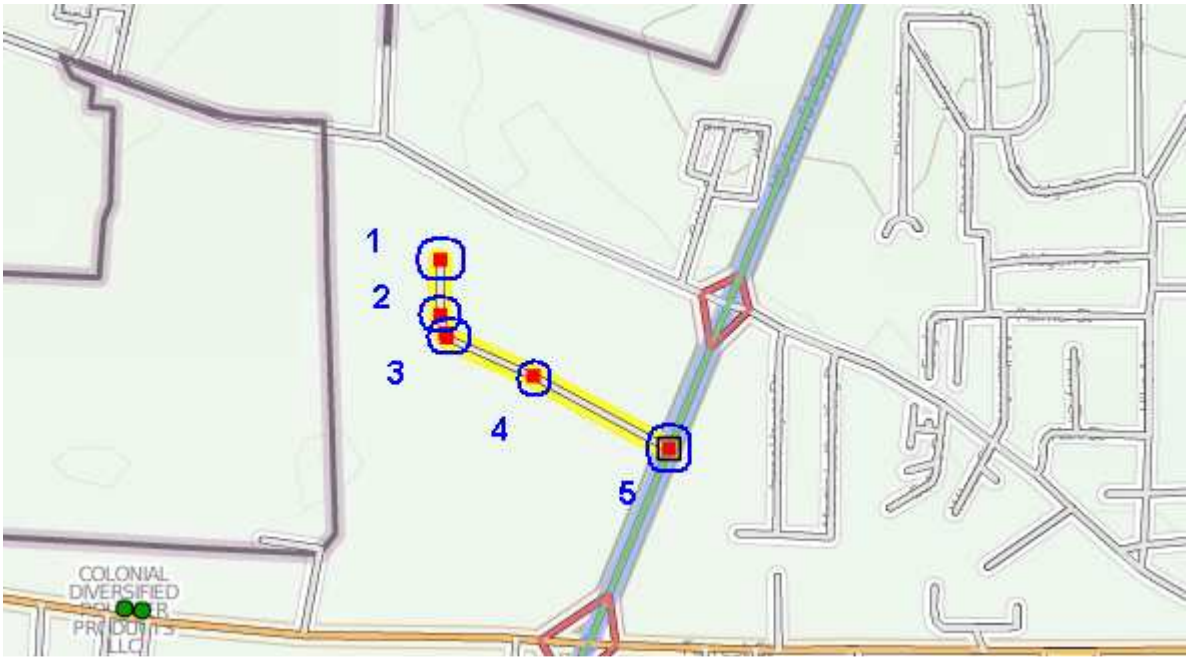

Figure 1: Example of a way

Figure 1 illustrates the difference between junction nodes and regular nodes. Nodes 2, 3 and 4 are regular nodes since they belong only to one street. Node 5 is a junction node since it is at the intersection of two streets so it belongs to two streets. Note that the first and the last node of a street are an exception; they will always be considered junction nodes for the purpose of this application. Therefore, node 1 is considered a junction node assuming it is the first node of the way.

To sum it up, the first node of a street, the last node of a street and any node that belongs to more than one street are junction nodes. Any node that belongs only to one street and is not the first or last node of that street is a regular node.

Links

A link, as defined in this application, is a relation that joins two consecutive junction nodes. Therefore, with this definition of links, streets as defined in OpenStreetMap are divided into multiple links.
Having said that, the first and the last node of a link are always junction nodes and in between there could be zero or more regular nodes.

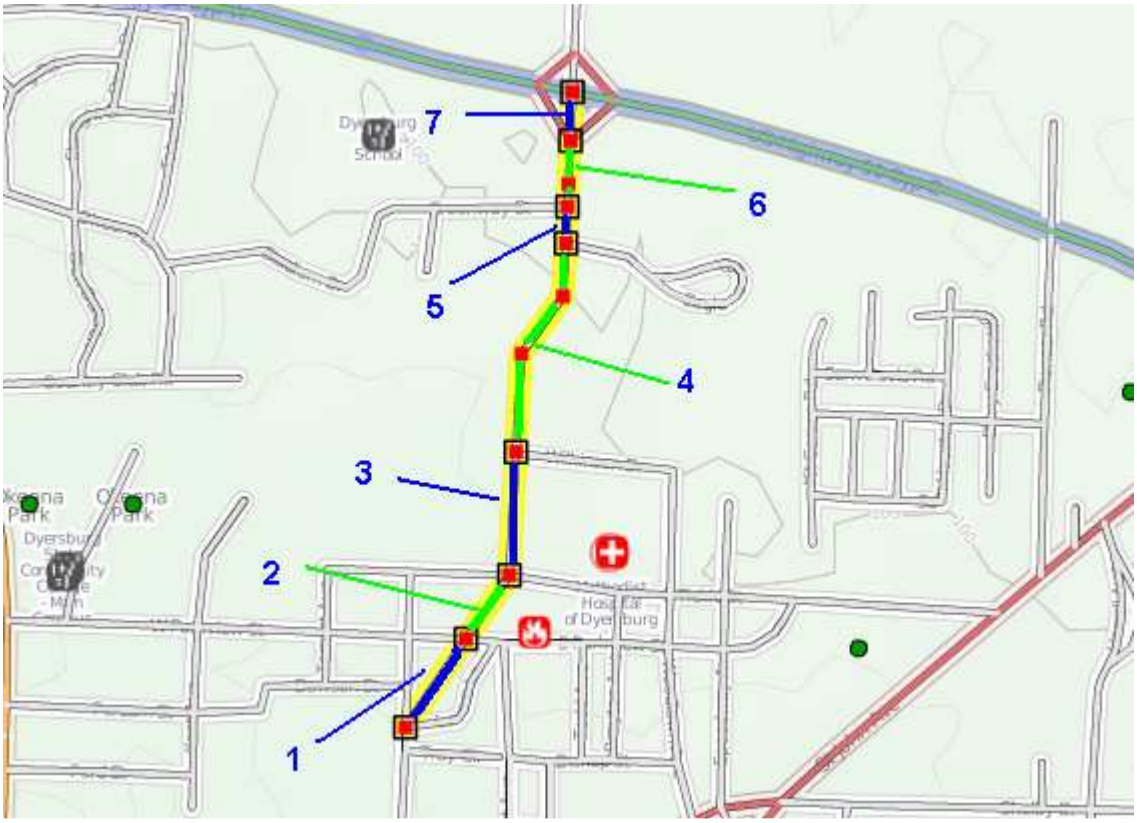

Figure 2: A Street divided into links

Using the definition of links, a street like the one in figure 2 will be divided into 7 links.
The junction nodes are the red dots with the black contour and the regular nodes are the red dots. For example link 4, has as starting and ending nodes two junction nodes and in between there are two regular nodes.

Overview

Here is a brief overview of the topics this report will cover:

The context section illustrates some information about the OpenStreetMap project and the OpenStreetMap API. Focus will be on the data and functions that are valuable for this application. There is also a brief overview of other technologies used in the application as RDF and RDF-S.

The System design section discusses three possible algorithms to solve the problem.
The first possible solution is to use the OSM API call bbox that returns all the nodes and the streets in a dimension specific bounding box.

The second possibility is a recursive algorithm that starts from the initial node and recursively makes new API calls to retrieve information about more nodes as long as these nodes are within the bounding box.
The third possible solution requires the download of an OpenStreetMap dump file that has all the information available in the OSM database.
Each solution is discussed in details listing its advantages and disadvantages and finally the recursive algorithm is chosen to implement the algorithm.

The system implementation section describes in details the chosen algorithm and the how it is implemented. This section contains a flowchart and a pseudo code that give a high level overview of the execution of the application. In this section there is also a description of the exploited OSM API calls used to retrieve the data. In addition, all the methods that are used in the code are listed here with a brief description of the functionality of each one of them.

The evaluation section assesses the correctness of the results generated by the application. An example illustrates with great details and step by step the execution of the algorithm starting from the initial and adding ways and nodes recursively. A visual simulation of the execution of the code is also provided to facilitate understanding the execution. Finally, the resulting RDF file is illustrated showing that it contains all the required data.

The last section concludes with the overall results of the application and the possible future works to improve the application and to add more functionalities to it.

2. Context

OpenStreetMap

OpenStreetMap is an open source map containing geographical information such as streets, nodes, monuments, practical information and so forth. Anyone can freely download and use the full information for any purpose they like under an open source license. Currently the most prominent use of OSM data is to render beautiful, rich maps such as the examples you can find on www.openstreetmap.org, but there are plentiful other applications too [1]. In OpenStreetMap any registered user could modify map information.
OpenStreetMap follows a similar concept as Wikipedia does, but for maps and other geographic facts. Users registered in OpenStreetMap gather geographical data from different sources and can upload it to the central database of OpenStreetMap where it could be seen, corrected and modified by other users.

The project was started because most maps you think of as free actually have legal or technical restrictions on their use, holding back people from using them in creative, productive, or unexpected ways. To foster these creative and unexpected uses, OpenStreetMap also does not limit the type of information people can add into the database, as long as it is factually correct, verifiable and does not infringe on anyone else's copyright. You never know for what interesting purposes it can be used for in future [7].

One could argue that geographical information found in such projects is not accurate. Here the argument is the same as for Wikipedia. There will be always users who would intentionally or unintentionally introduce inaccurate data. However, since the project is open source other users could see the erroneous data and would hopefully correct the errors.

There are three key elements that make up the map data for OpenStreetMap namely nodes, ways and relations [1].

Node

A node is the basic element of the OpenStreetMap scheme. Nodes consist of latitude and longitude (a single geospatial point). Nodes are needed to define a way, but a node can also be a standalone unconnected point representing something like a telephone box, a pub, a "place" name label, or all kinds of other points of interest (POI) [2].

Way

A way is an ordered interconnection of at least 2 and at most 2000 nodes that describe a linear feature such as a street, or similar. Nodes can be members of multiple ways.

Relation

A relation can group other elements together, nodes, ways, and maybe even other relations. Elements are 'members' of the relation, and each membership has a 'role'. As with other types of elements, a relation may have an arbitrary number of tags. You may also have duplicate nodes, ways or relations within a single relation.

In this project, the two elements that interest us are ways and nodes.

Given a starting node, latitude and longitude, the goal is to retrieve all the streets inside the bounding box where the starting node is in the center of the bounding box.

OpenStreetMap API

This API is based on the ideas of the RESTful API.
REST-style architectures consist of **clients** and **servers.** Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of "representations" of "resources". A resource can be essentially any coherent and meaningful concept that may be addressed. [5]

The API is the server component to which REST requests are addressed. The REST requests take the form of HTTP GET, PUT, POST, and DELETE messages. The API is currently accessible using the following URL: http://api.openstreetmap.org/

In this application the client is the java application that initiates requests to the OSM API. The REST requests take the form of HTTP GET, PUT, POST, and DELETE messages. Only GET requests are used since the purpose of the application is only to retrieve information from OpenStreetMap.

RDF (Resource Description Framework)

RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed [3]. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

RDF is designed to be read and understood by computers and not designed to be displayed to humans [6]. RDF documents are written in XML. The XML language used by RDF is called RDF/XML. By using XML, RDF information can easily be exchanged between different types of computers using different types of operating systems and application languages. RDF identifies things using Web identifiers

(URIs), and describes resources with properties and property values. A resource is anything that can have a URI, such as http://www.w3schools.com/rdf or in this application it could be for example a link. The property describes the resource such as the name of the link. The property value is the value of a property such as "17th Street" which is the name of the link.
In this application the end result is an RDF/XML output file containing the information retrieved by the algorithm.

RDF-S (RDF Schema)

To support the sharing and reuse of formally represented knowledge among Artificial Intelligence systems, it is useful to define the common vocabulary in which shared knowledge is represented. A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called ontology[9].
RDF Schema is an extensible knowledge representation language, providing basic elements for the description of ontologies.
RDF describes resources with classes, properties, and values. In addition, RDF needs a way to define application-specific classes and properties. Application-specific classes and properties must be defined using RDF Schema.

An Ontology to describe streets

Below is the ontology that describes the main classes and properties found in the resulting RDF response of the application.

The class *node* represents nodes and has two properties *gbLat* (latitude) and *gbLong* (Longitude) which describe the node. The class *link* represents the links and it has as properties *hasFrom* that indicates the starting node of the link, *hasTo* that indicates the ending node, *hasEdge* that indicates the regular nodes in between the starting and the ending node and finally *hasLength* that indicates the length of the link [10].

```
@prefix upf: <http://larkc.cefriel.it/ontologies/urbanpathfinding#>.
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

upf:Node a rdfs:Class;
rdfs:label "Node";
rdfs:comment "Conjunction of two or more links".

upf:gbLat a rdf:Property;
rdfs:range xsd:double;
rdfs:label "lat (gauss-boaga)";
rdfs:comment "Latitude of a geographical point expressed in Gauss Boaga reference system".

upf:gbLong a rdf:Property;
rdfs:range xsd:double;
rdfs:label "long (gauss-boaga)";
rdfs:comment "Longitude of a geographical point expressed in Gauss Boaga reference system".

upf:Link a rdfs:Class;
rdfs:label "Link";
rdfs:comment "Pieces of roads (oriented according to the direction)".

upf:hasFrom a rdf:Property;
rdfs:label "from";
rdfs:comment "Relation between a link and its start node";
rdfs:domain upf:Link;
rdfs:range upf:Node.

upf:hasTo a rdf:Property;
rdfs:label "to";
rdfs:comment "Relation between a link and its end node";
rdfs:domain upf:Link;
rdfs:range upf:Node.

upf:hasEdge a rdf:Property;
rdfs:label "has edge";
rdfs:comment "Relation between a link and its edge nodes";
rdfs:domain upf:Link;
rdfs:range upf:Node.

upf:hasLength a rdf:Property;
rdfs:label "length";
rdfs:comment "Length of a link";
rdfs:domain upf:Link;
rdfs:range xsd:float.
```

```
upf:hasStart a rdf:Property;
rdfs:label "start";
rdfs:comment "The start point of the path";
rdfs:domain upf:Path;
rdfs:range upf:Node.

upf:Street a rdfs:Class;
rdfs:label "Street";
rdfs:comment "Steets".

upf:streetName a rdf:Property;
rdfs:domain upf:Street;
rdfs:range xsd:string;
rdfs:label "street name";
rdfs:comment "the name of the street".
```

3. System Design

Three possible solutions to solve the problem are described.

First solution

One solution is to use the bbox method. It is a function in the OpenStreetMap API that gets all the elements inside a bounding box defined by the user.

```
GET /api/0.6/map?bbox=left,bottom,right,top
```

This function takes 4 arguments. The bottom longitude, the top longitude, the left latitude and the right latitude and it returns all the nodes and streets in that area. However, this function does not apply recursively.

In order to use this API function, the coordinates of the area of interest must be known in advance and this is not the case in this application. Only two inputs are known a-priori: the starting node and the dimension of the area around the starting node.

Second Solution

The second possible solution is a recursive algorithm that makes several API calls recursively and retrieves the elements from inside the box.
The algorithm takes as input the initial node that is at the center of the bounding box and the dimension of the box. The streets that traverse the initial nodes are retrieved with all the nodes that make up those streets. Then for each node, if it is inside the bounding box then it is added to the list of nodes and new API calls are made to retrieve information about the streets that traverse that node. The algorithm keeps iterating until all the elements inside the bounding box are retrieved.

Third Solution

Planet.osm is a snapshot of the OpenStreetMap database. It is the latest revision of the nodes, ways, relations and changesets in the database. This of course includes all the tags. A new version of planet.osm is released weekly. The current size of a planet.osm file is over 160 gigabytes (reduced to 7.3GB with bzip2 compression) as of November 2009. [4]
The third possible solution is to use the planet.osm to retrieve the needed information.

One advantage of using the planet dump is the reduction of the number of API calls which are slow and depend heavily on the internet connection. Since the planet

dump could be locally stored on a machine, then the application could be used offline since all the data would be retrieved from the dump file.

However, the major drawback is that the planet.osm file needs to be updated weekly to ensure up to date information. Even if the planet.osm is downloaded each week some information at the time of use by the application might have become out of date due to an update that was not captured in the last dump file.
Another disadvantage is the large size of the dump file. Most potential users of the application wouldn't be willing to store a 160 Gb file locally on their machine.

Given the system requirements and taking into consideration the advantages and disadvantages of each possible solution, the second solution (recursive algorithm) is chosen to extract the streets. This solution will always yield results that are update since it will make live API calls to retrieve the data, it will not require a lot of disk usage since no OpenStreetMap data is stored on the client side and finally streets will be extracted recursively starting from the center of the bounding box and moving toward the edges.

4. System Implementation

At a high level, the methods that implement the functionality of the code are divided into three groups according to their function.
There are the methods that communicate with the OpenStreetMap API, the methods that perform the recursive calculation, and the methods that rearrange the results and write them as an RDF file.

In the first group there are the functions that read the input arguments and prepare the web service calls to be made to the API using the HttpClient library. Once the webservice calls are made, responses are written to a file to be parsed later on in order to retrieve the needed information.
Once the first xml response is available, starting from the initial node a recursive algorithm evaluates the responses and decides accordingly whether more web service calls shall be made to retrieve new streets and nodes. All the streets and nodes that are found within the bounding box are added to a list array.
Finally, once the list of all the elements inside the bounding box is populated, a function performs the division of streets into links and writes the response as an RDF file according to a specific schema.

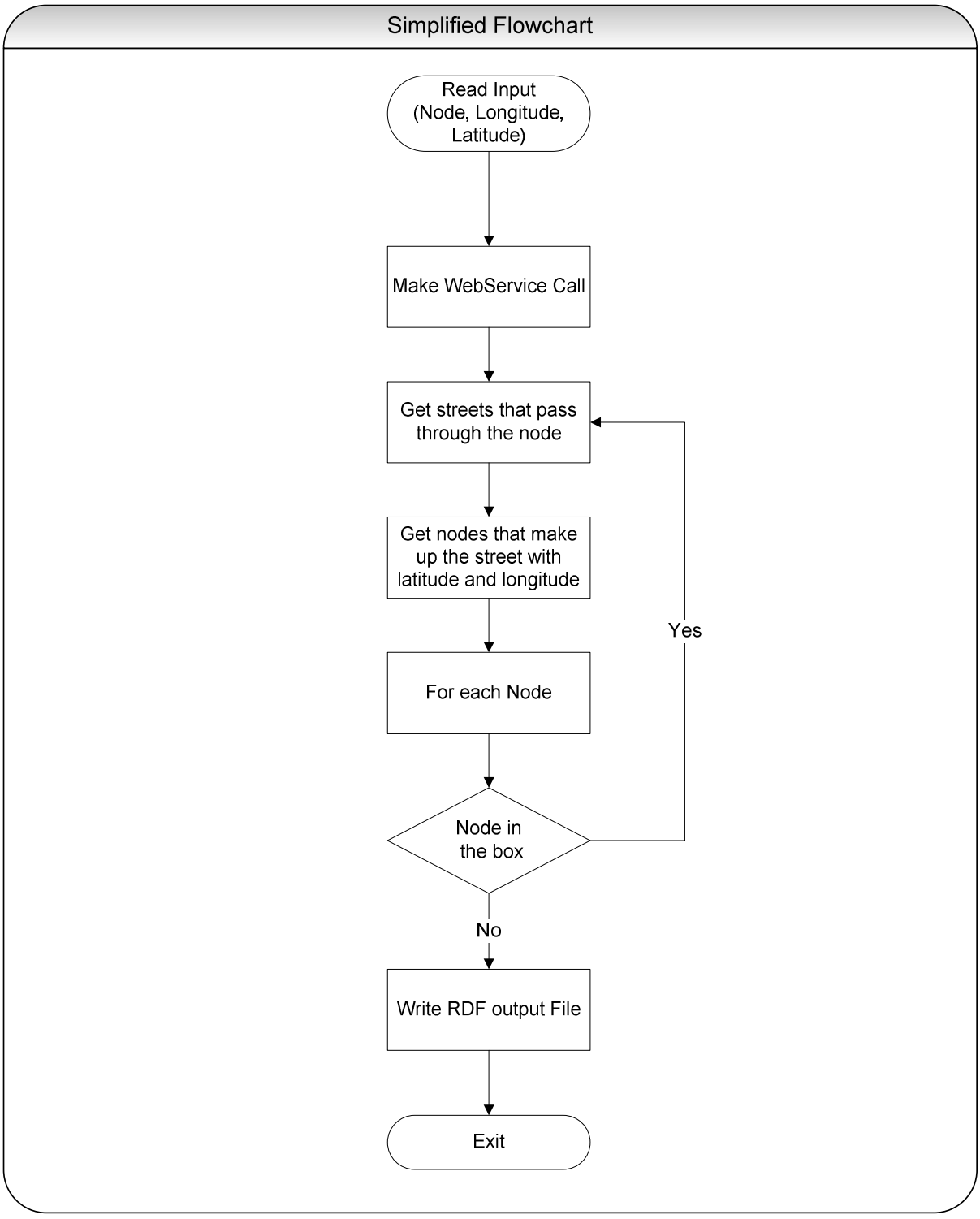

Figure 3: Simplified Flowchart

Pseudo Code

The following pseudo code illustrates how the application runs:

```
1   Read the starting Node and the dimension of the bounding box in Longitude and
2   Latitude
3   Retrieve the streets that pass through the initial node by calling the function
4   /api/0.6/node/#id/ways
5
6   Initialize the list of StreetStructs (street objects). Each StreetStruct contains
7   information about the street and a list of all the nodes that belong to that street.
8
9   Loop through the list of street.
10  For each street, loop through its list of nodes
11  If node is not a junction node, skip node.
12  If node is a junction node
13  Check whether it is within the bounding box.
14  If a node is a junction node and the streets that pass through it are not already
15  in the street list.
16  Create new StreetStruct and add it to the street list.
17
18  For each street in the list
19  Break up the street into Links
20  Write the RDF file.
21  Exit
```

OpenStreetMap API Calls

Two OpenStreetMap API calls are used in this application.
`GET /api/0.6/node/#id/ways` returns all the ways passing through the specified node.

Example:

http://api.openstreetmap.org/api/0.6/node/203060671/ways

```
<osm version="0.6" generator="OpenStreetMap server">
<way id="19546565" visible="true" timestamp="2008-01-02T11:59:12Z" version="1"
changeset="511198" user="DaveHansenTiger" uid="7168">
<nd ref="203060665"/>
<nd ref="203060668"/>
<nd ref="202841203"/>
<nd ref="203060671"/>
<nd ref="202857573"/>
<nd ref="203060673"/>
<nd ref="202925724"/>
<tag k="highway" v="residential"/>
<tag k="name" v="20th St"/>
<tag k="tiger:cfcc" v="A41"/>
<tag k="tiger:county" v="Knox, TN"/>
<tag k="tiger:name_base" v="20th"/>
<tag k="tiger:name_type" v="St"/>
<tag k="tiger:reviewed" v="no"/>
<tag k="tiger:separated" v="no"/>
<tag k="tiger:source" v="tiger_import_dch_v0.6_20070829"/>
<tag k="tiger:tlid" v="42620102:42620104:42620114:42620116:42620072"/>
<tag k="tiger:upload_uuid" v="bulk_upload.pl-f459fb5c-e923-4061-be5e-
82c37c53f049"/>
<tag k="tiger:zip_left" v="37916"/>
<tag k="tiger:zip_right" v="37916"/>
</way>
<way id="19548039" visible="true" timestamp="2008-01-02T12:06:04Z" version="1"
changeset="511198" user="DaveHansenTiger" uid="7168">
……
</way>
</osm>
```

`GET /api/0.6/[node|way|relation]/#id` returns an XML representation of the element containing the longitude and latitude of the node.

Example:

http://api.openstreetmap.org/api/0.6/node/203060671/

```
<osm version="0.6" generator="OpenStreetMap server">
<node id="203060671" lat="35.9574371" lon="-83.9380504" version="6"
changeset="4514154" user="Rick Presley" uid="182" visible="true"
timestamp="2010-04-24T18:13:35Z"/>
</osm>
```

Functions and Methods

This section lists all the functions in the code and gives a brief description of their functionality.

public class NodeStruct

The class NodeStruct is used to store information about nodes. It holds the ID of the node, its longitude, its latitude, the number of streets that intersects at this node, and a StreetStruct array of those streets that intersect at the node.

public class StreetStruct

The class StreetStruct is used to store information about streets (ways). Each StreetStruct holds the ID of the street, the street name, the type of street whether it is a one way or a two way street, and a NodeStruct array of the nodes that are part of that street.

public String ReadInputID()

The ReadInput() method reads the ID the starting node from standard input. The method verifies that the entered starting node contains only numbers and exits in case letters were found. The method doesn't check if the node ID exists in the OpenStreetMap database.

public double GetInputLongitude()

The GetInputLongitude() reads the longitude of the bounding box from the standard input. The method doesn't perform error checking.

public double GetInputLatitude()

The GetInputLatitude() reads the laltitude of the bounding box from the standard input. The method doesn't perform error checking.

public String CreateWayURL(String nodeID)

The CreateWayURL(String nodeID) creates a URL of the form `http://api.openstreetmap.org/api/0.6/node/nodeID/ways` that will be sent to the OpenStreetMap API as HTTP GET request and will return the streets that traverse this node as xml elements <ways> and each street element will contain the nodes <nd> that make up the street as children of the <way> element.

public String CreateNodeURL(String nodeId)

The CreateNodeURL(String nodeID) creates a URL of the form `http://api.openstreetmap.org/api/0.6/node/nodeID` that will be sent to the OpenStreetMap API as HTTP GET request and will return information about that specific node. In this application we are mostly interested in the longitude and latitude of the node.

public String GetResponse(String url)

The GetResponse(String url) takes as arguments the URL created by CreateNodeURL() or CreateWayURL() functions and uses the java HttpClient library to query the OpenStreetMap API. The returned xml response is written to a file to be parsed later on in the application.

public ArrayList ParseXmlResponse(String filename)

The ArrayList ParseXmlResponse(String filename) takes as input the xml response file returned by the GetResponse() method. The method parses the xml file and stores ways in an array of StreetStructs, for each way it stores its ID, name and type.

Moreover, it creates an array of NodeStructs, containing all the nodes (with the information about the node) for that street, which will be inserted in the StreetStrcut.

public NodeStruct getLonLat(String NodeID)

The getLonLat(String NodeID) parses the xml response returned by the API call `http://api.openstreetmap.org/api/0.6/node/nodeID` and retrieves the longitude and the latitude of the node which will be used later to determine whether new nodes are inside the bounding box.

public NodeStruct getStreetInfo(String nodeID)

The getStreetInfo(String nodeID) is used for the sole purpose of counting the number of streets that pass through a specific node.

public boolean containsNode(NodeStruct n, ArrayList list)

The containsNode(NodeStruct n, ArrayList list) method takes as arguments a NodeStruct and an array of NodeStructs and it traverses the array list checking whether the nodeStruct n is already in the array. This method comes in handy during the recursive algorithm in order not to enter duplicate nodes when nodes are being added to the array of nodes.

public boolean containsStreet(StreetStruct S, ArrayList list)

The containsStreet(StreetStruct S, ArrayList list) method is similar to the ContainsNode(), but in this case, it ensures that no duplicate StreetStructs are introduced in the array of streets.

public void xmlWriter(ArrayList osmStreetList)

The xmlWriter(ArrayList osmStreetList) is the last method is to be called after the algorithm finishes its execution. The method takes as argument the final array of StreetStructs osmStreetList which will at the end of the execution contain information about all the streets and nodes in the bounding box initially specified.
The function will create the RDF file that will have all those information in it. The method will evaluate which nodes are junction nodes (nodes where streets intersect). It will also divide streets into links (A link is a connection between two junction nodes).

5. Evaluation

Many tests were performed to evaluate the accuracy of the returned results.
Here an example illustrates in details the execution of the algorithm and the obtained results.

Example

This example is used to illustrate the accuracy of the results generated by the algorithm highlighting the recursive retrieval of the streets.

The starting node is **203009666** and the dimension of the bounding box is **0.0066** in longitude and **0.0066** in latitude.

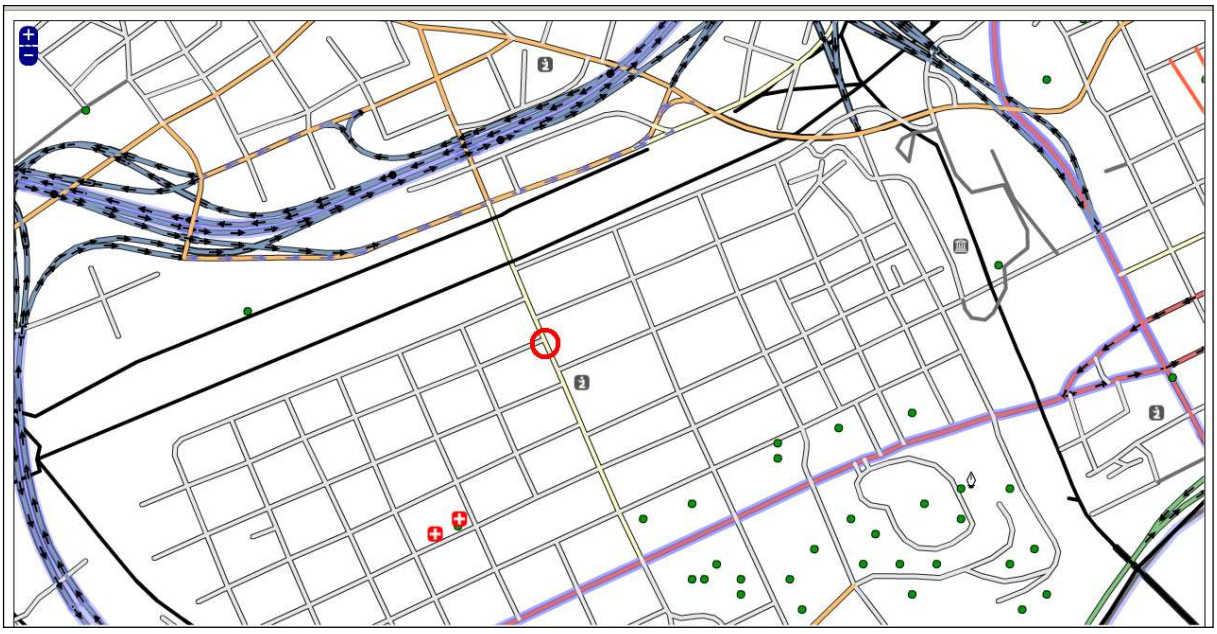

Figure 4: Initial Node

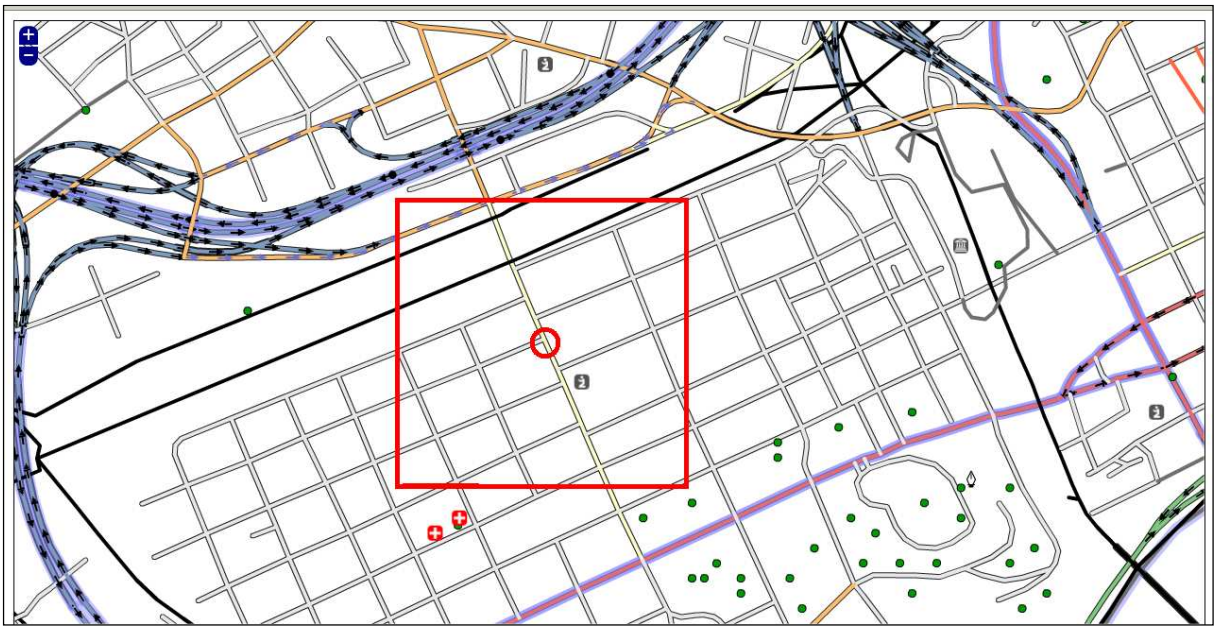

Figure 5: Initial Node and Bounding Box

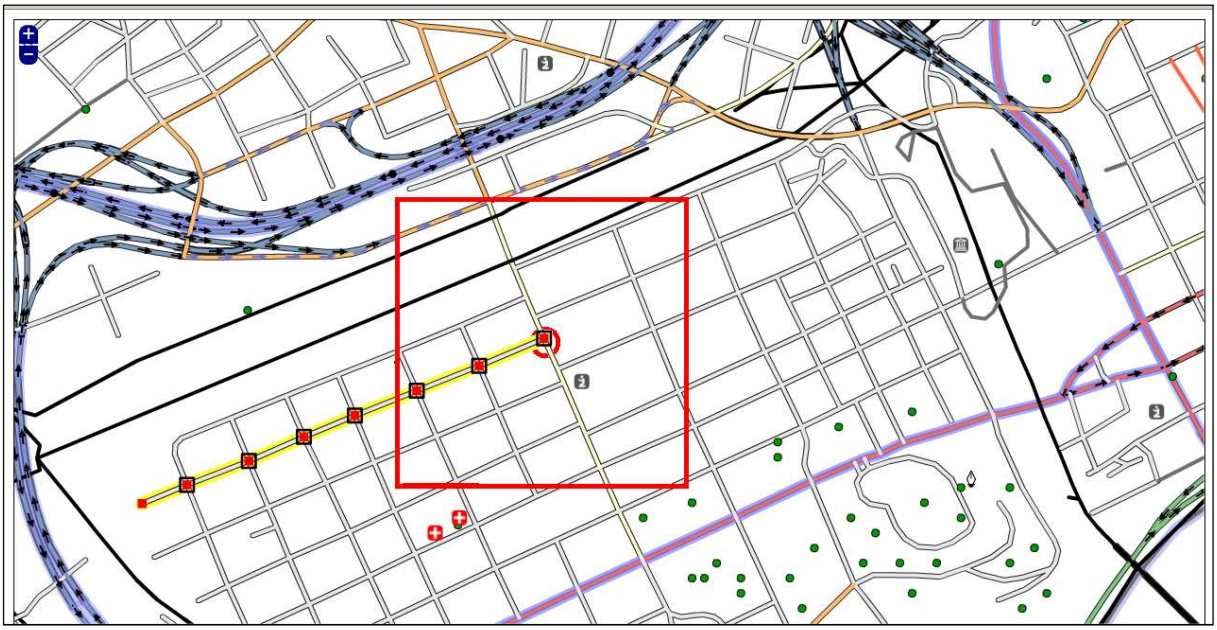

Figure 6: Adding 1st Street (Forest Avenue)

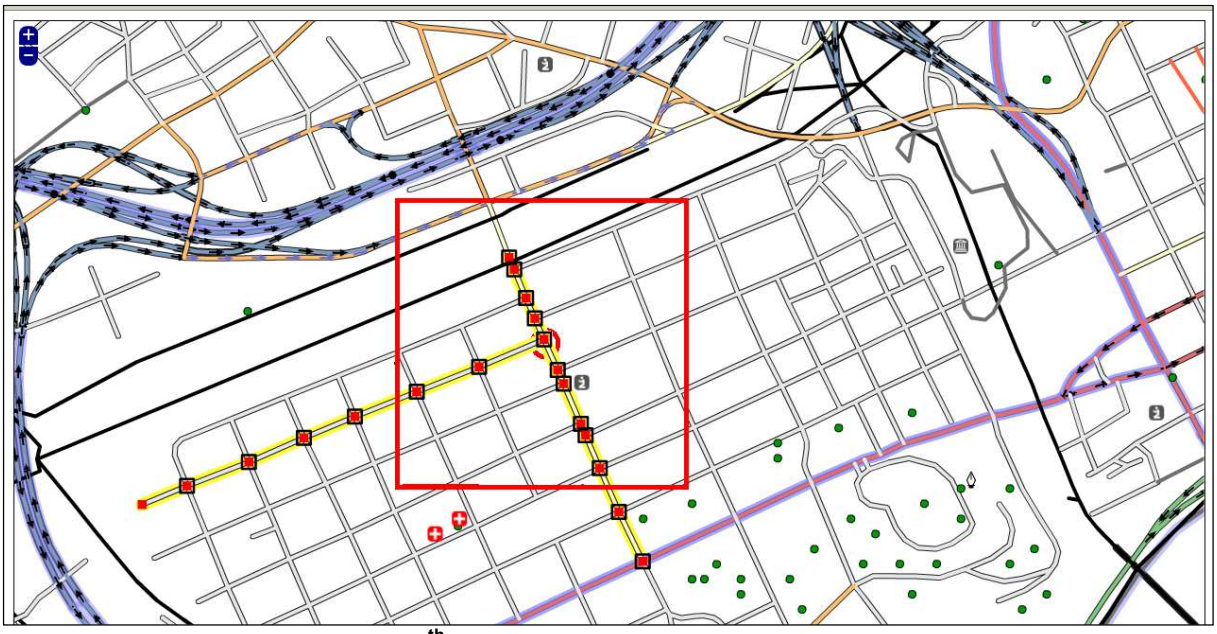

Figure 7: Adding 2nd Street (S. 17th Street)

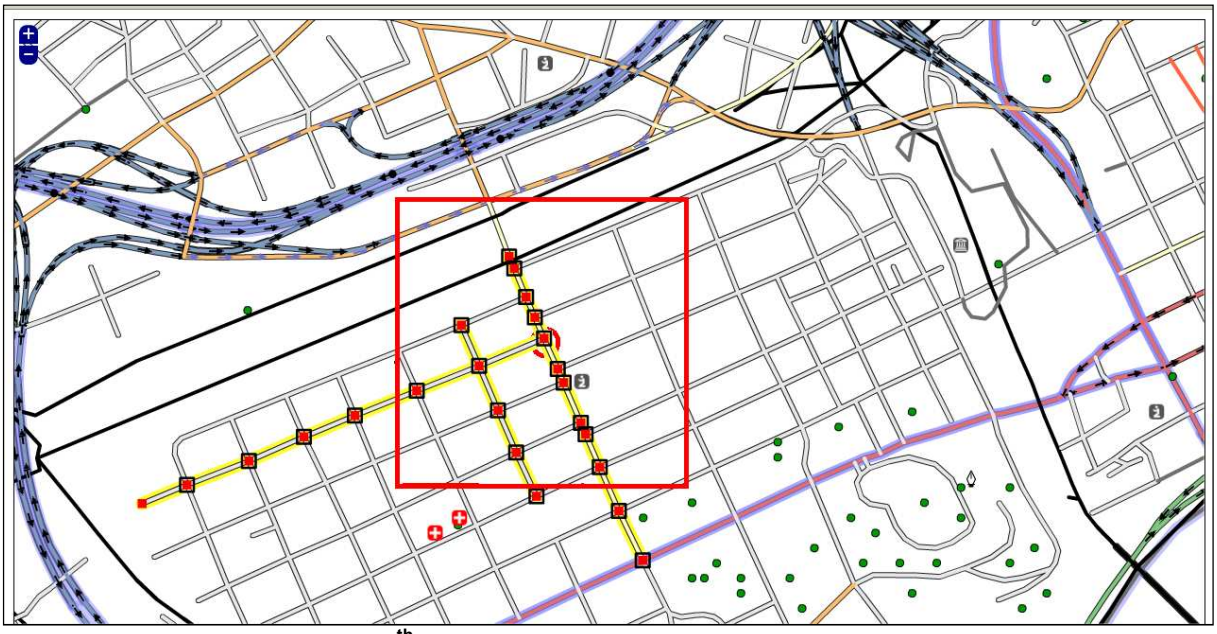

Figure 8: Adding 3rd Street (18th Street)

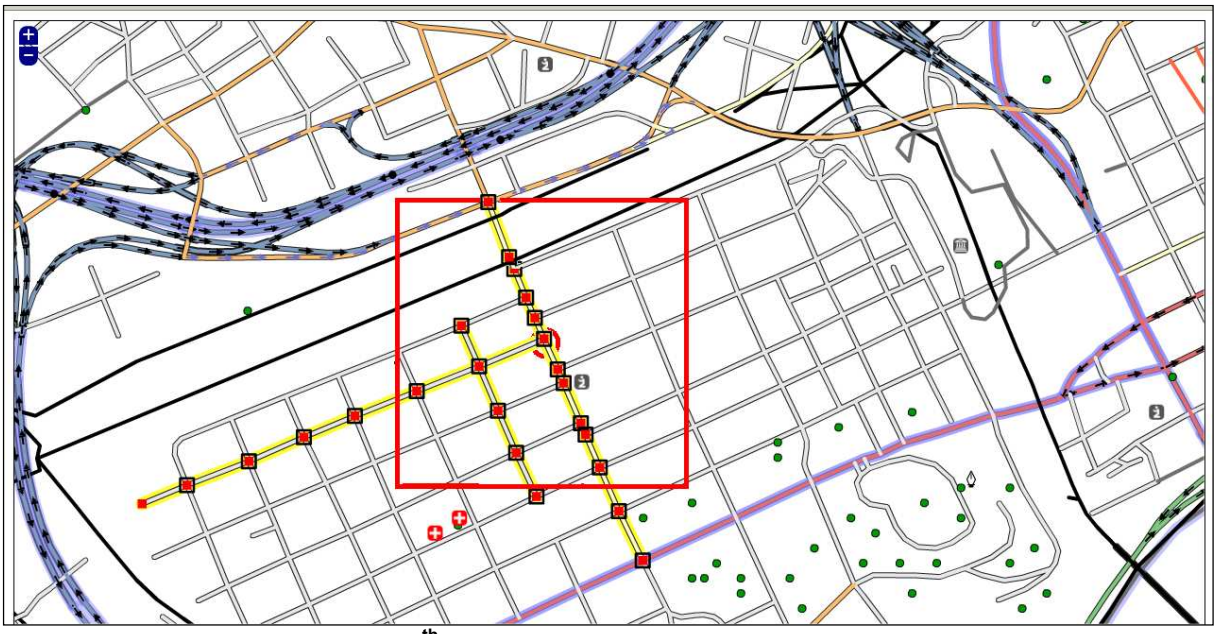

Figure 9: Adding 4th Street (N 17th Street)

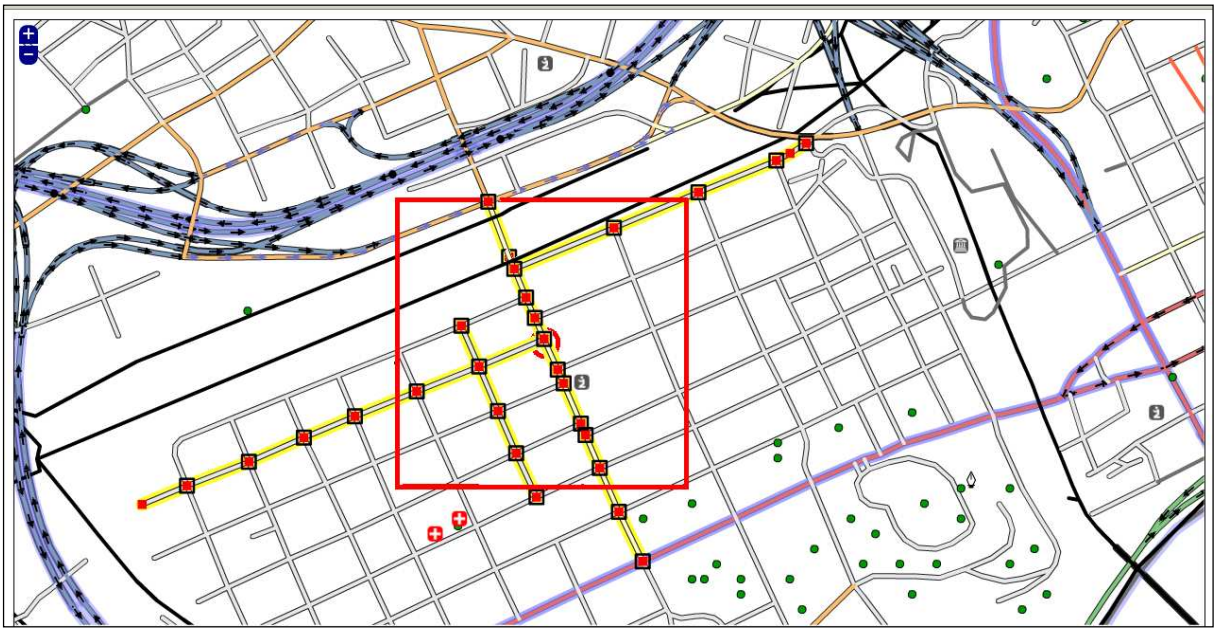

Figure 10: Adding 5th Street (Grand Avenue SW)

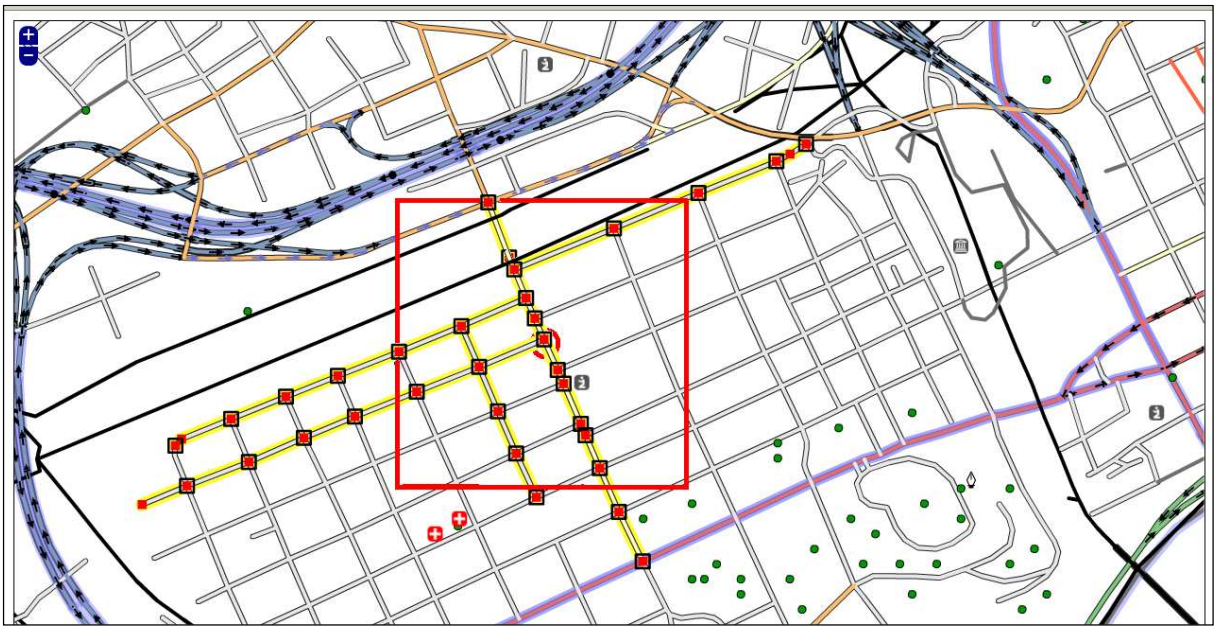

Figure 11: Adding 6th Street (Grand Avenue SW)

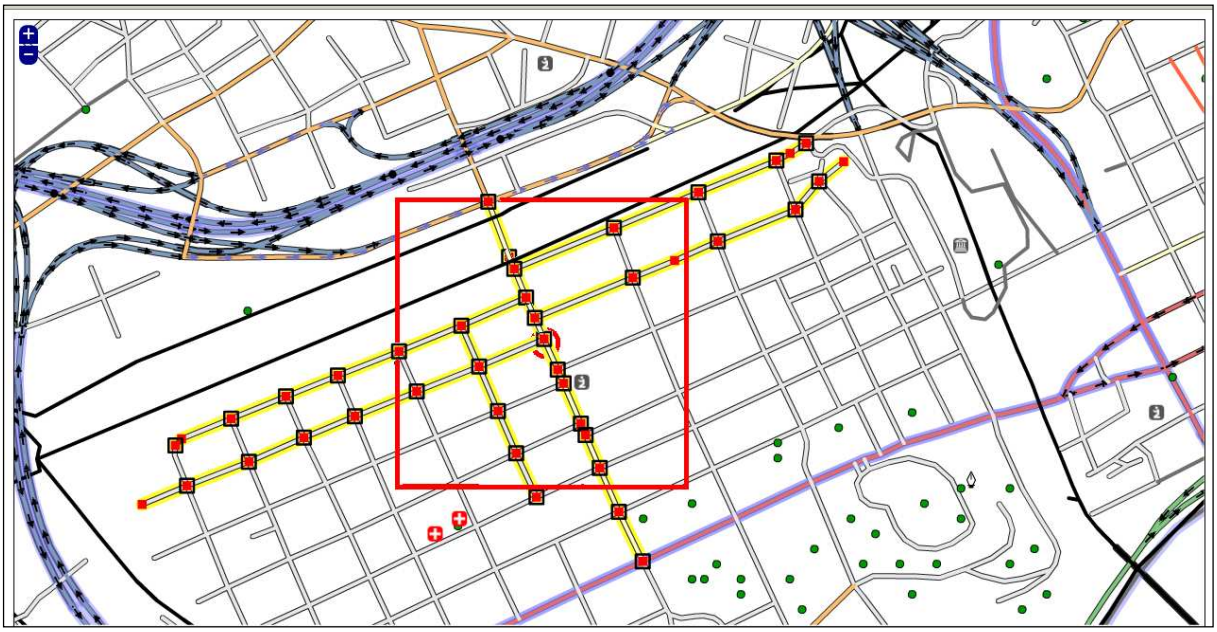

Figure 12: Adding 7th Street (Forest Avenue)

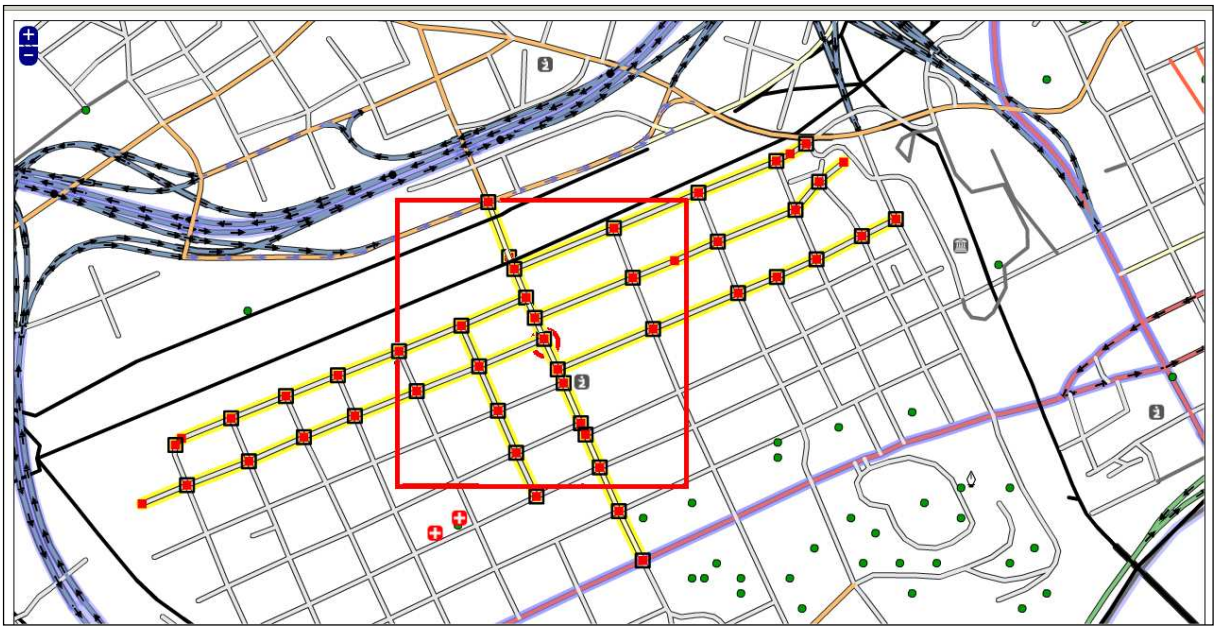

Figure 13: Adding 8th Street (Highland Avenue SW)

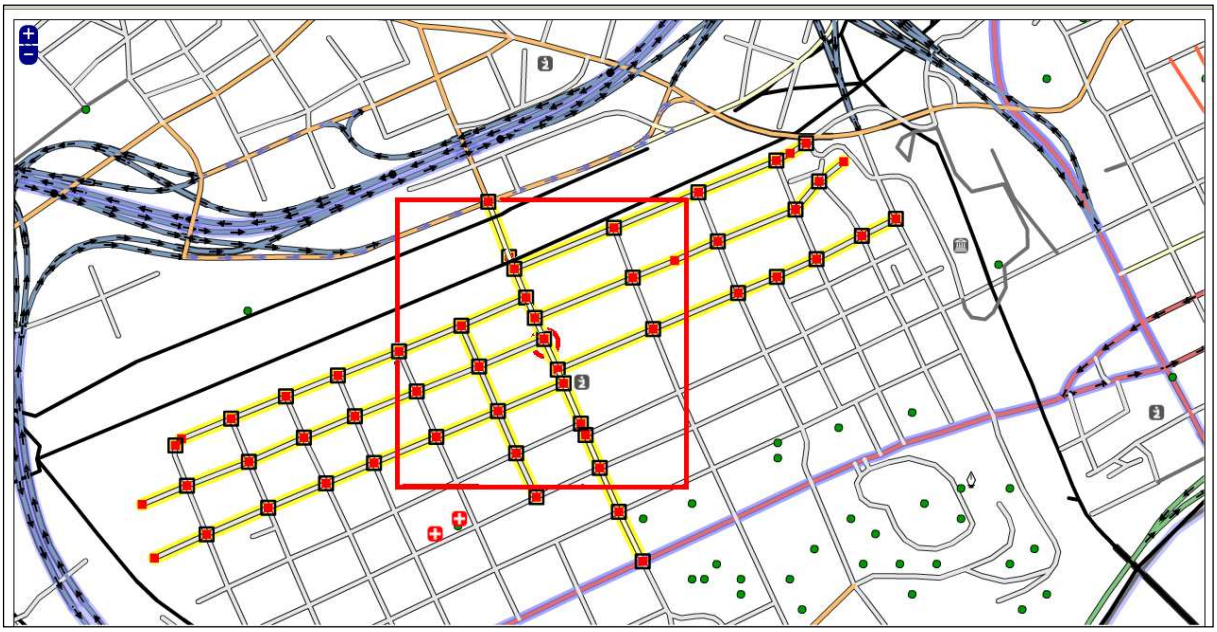

Figure 14: Adding 9th Street (Highland Avenue SW)

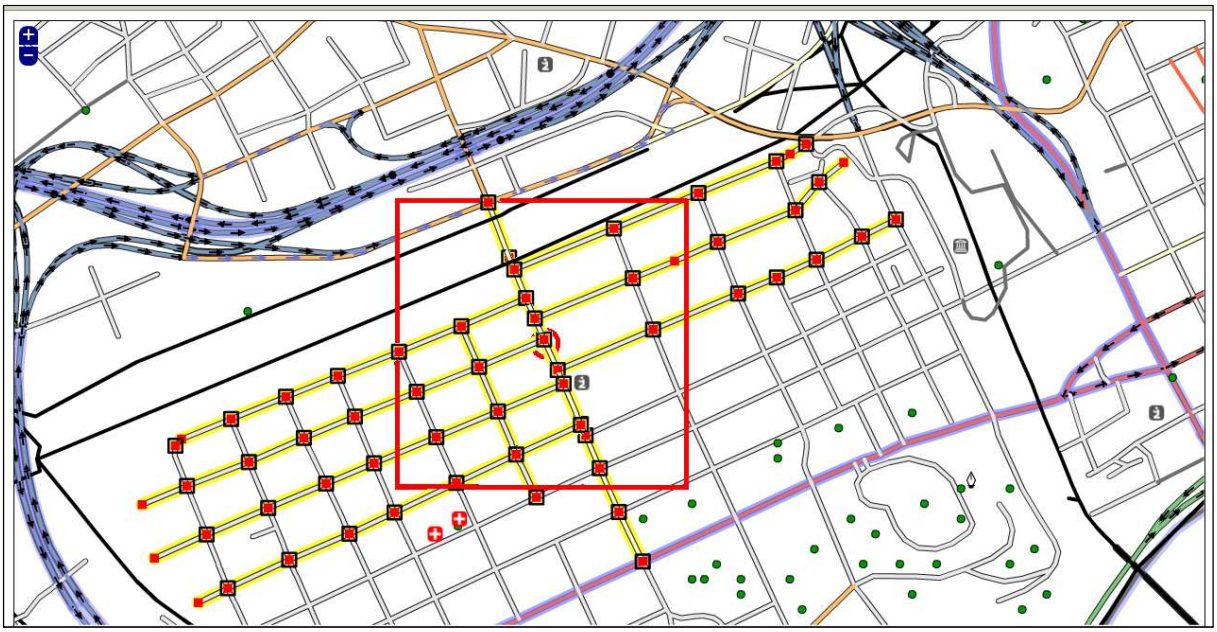

Figure 15: Adding 10th Street (Laurel Avenue)

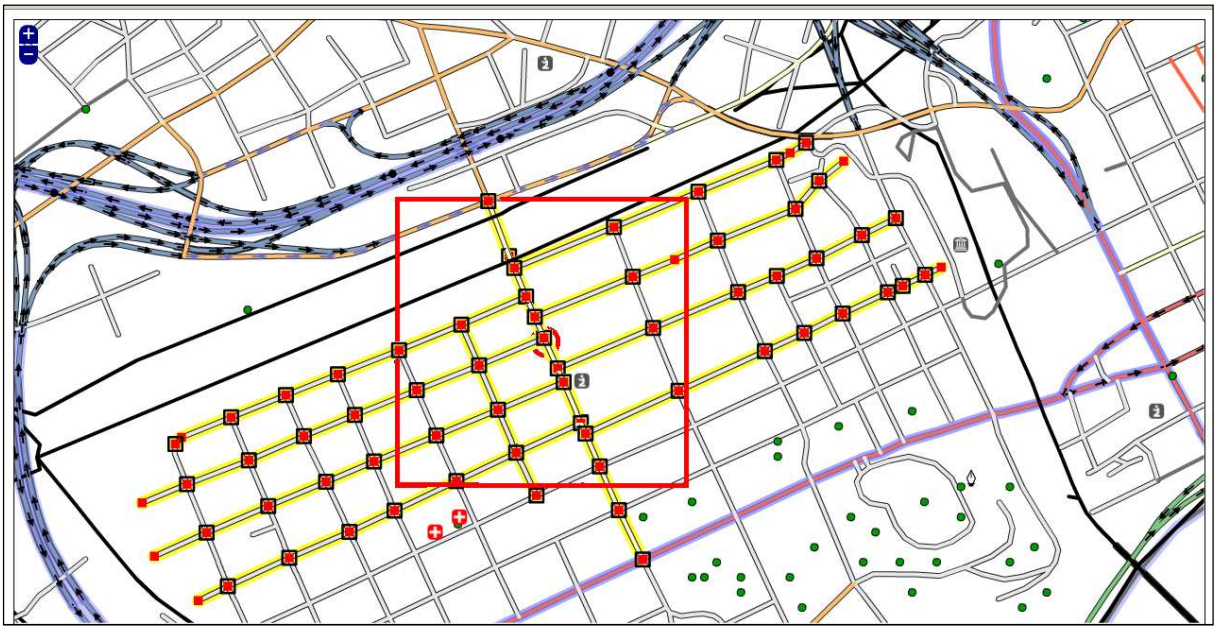

Figure 16: Adding 11th Street (Laurel Avenue)

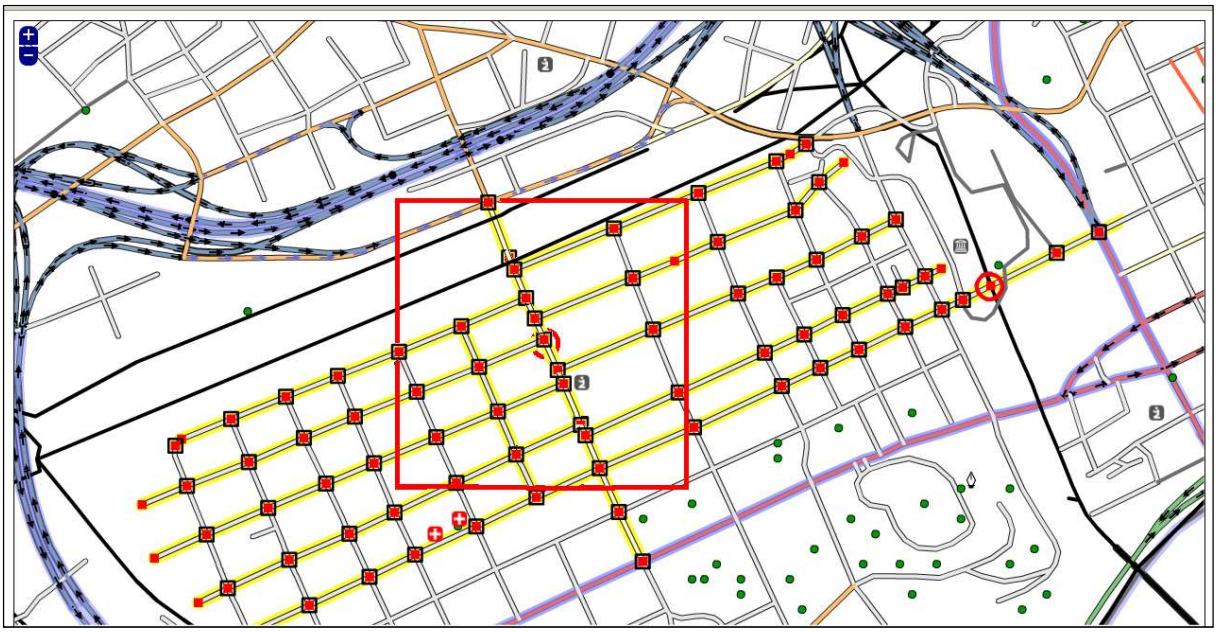

Figure 17: Adding 12th Street (Clinch Avenue SW)

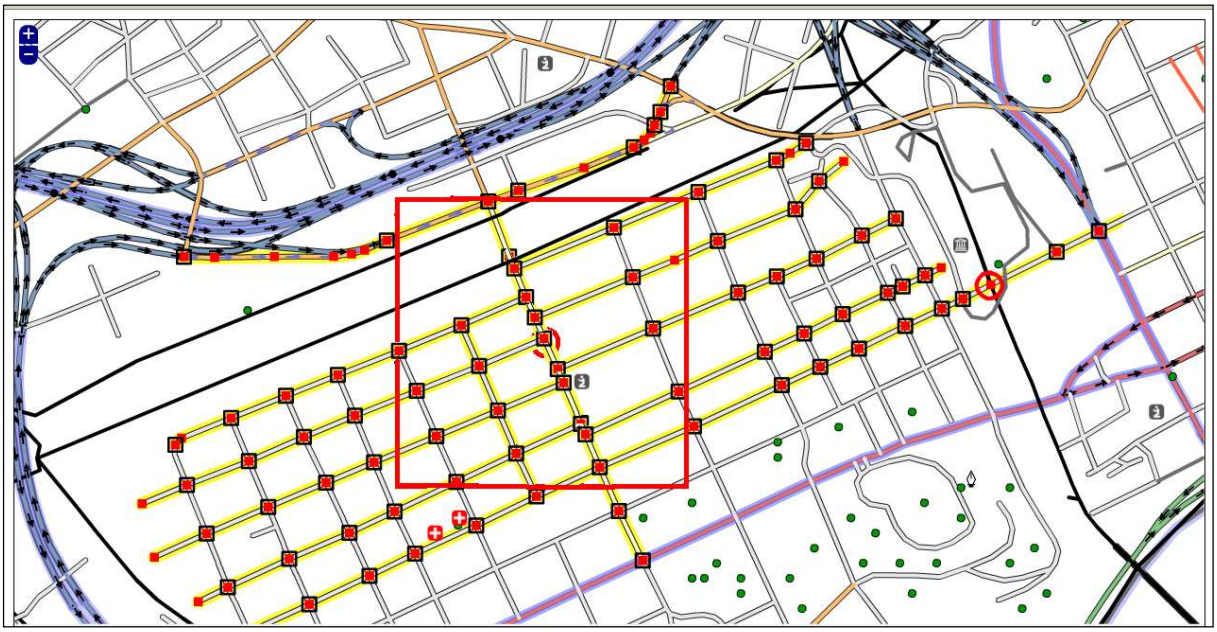

Figure 18: Adding 13th Street (Dale Avenue NW)

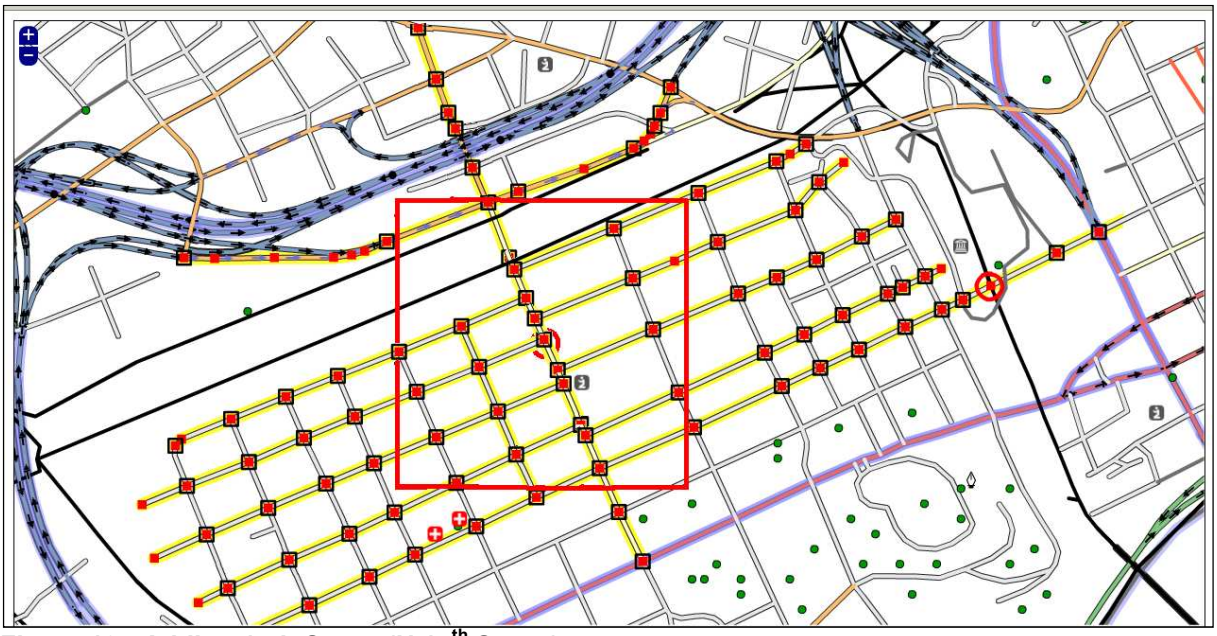

Figure 19: Adding 14th Street (N 17th Street)

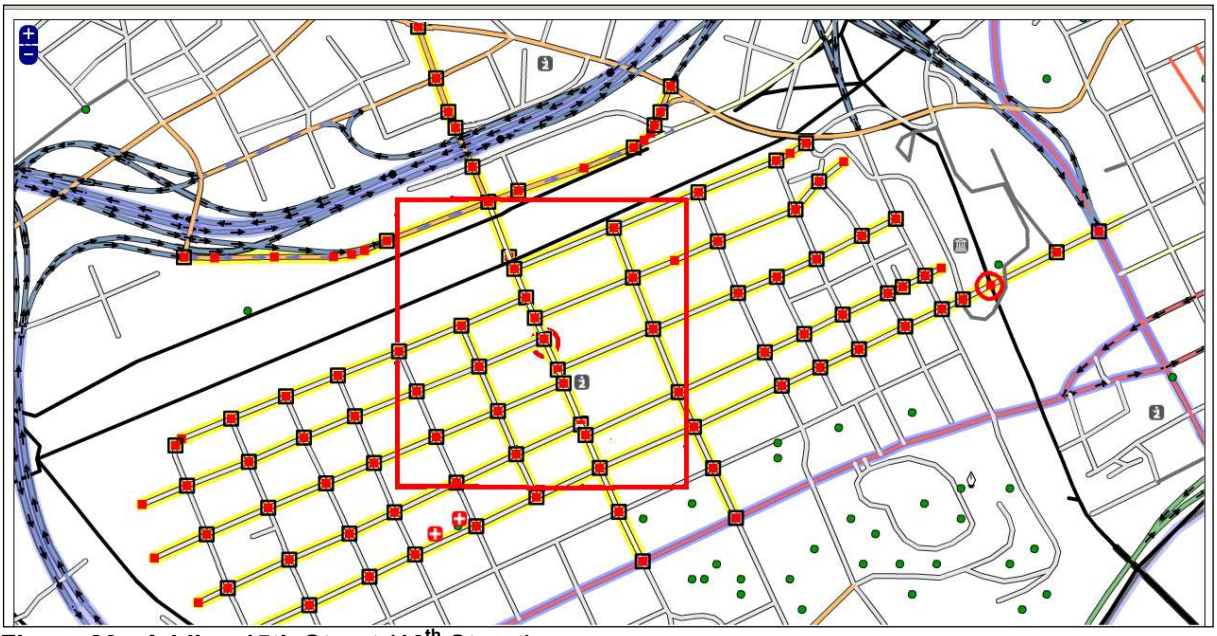

Figure 20: Adding 15th Street (16th Street)

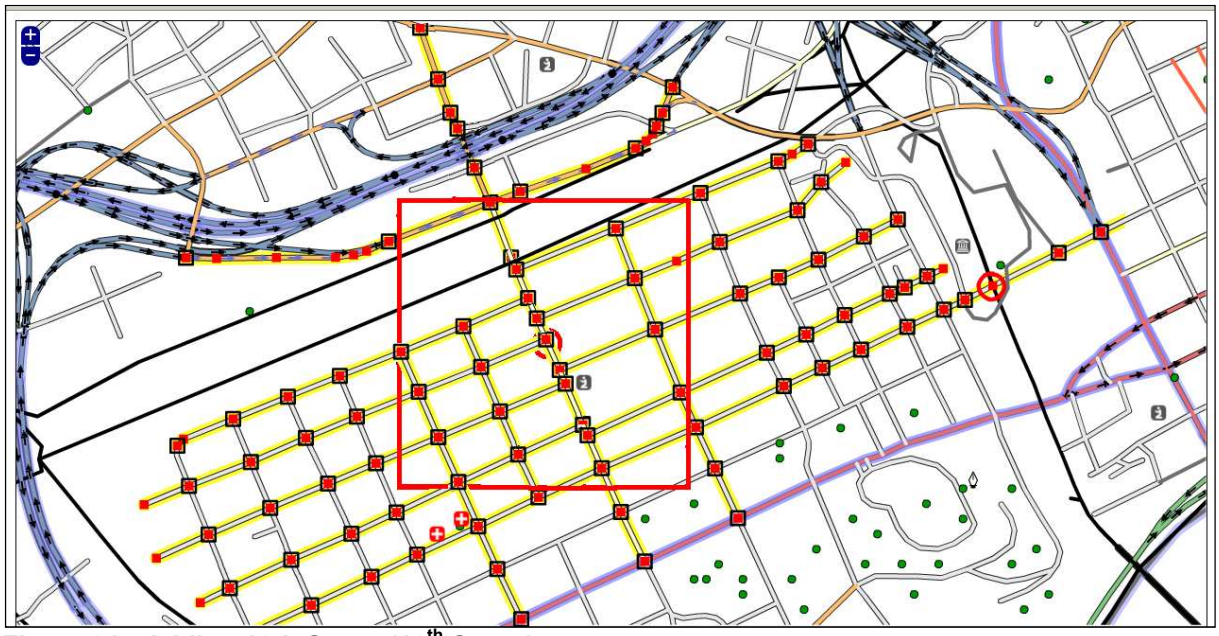

Figure 21: Adding 16th Street (19th Street)

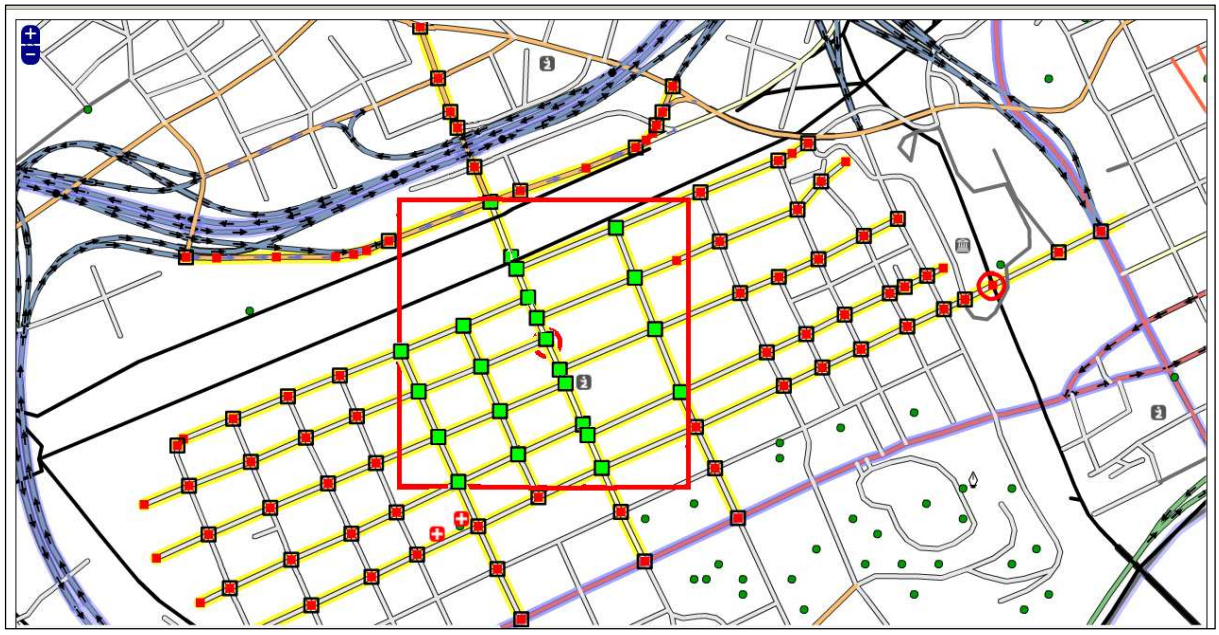

Figure 22: Final Result all street with at least one node in the bounding box are selected

RDF response of the example

Below is a part of the RDF response returned after the execution of example above.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xml:base="http://seip.cefriel.it/ama/resource/"
   xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
   xmlns:lud="http://www.linkingurbandata.org/onto/ama/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
   <rdf:Description rdf:about="links/link19548537L1">
      <lud:partOf rdf:resource="street/street19548537"/>
      <rdfs:label>Forest Ave</rdfs:label>
      <lud:lFrom rdf:resource="nodes/node203009666"/>
      <lud:lTo rdf:resource="nodes/node202832303"/>
      <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
      <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
      <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
   </rdf:Description>
   <rdf:Description rdf:about="links/link19548537L2">
      <lud:partOf rdf:resource="street/street19548537"/>
      <rdfs:label>Forest Ave</rdfs:label>
      <lud:lFrom rdf:resource="nodes/node202832303"/>
      <lud:lTo rdf:resource="nodes/node202932241"/>
      <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
      <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
      <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
   </rdf:Description>
   <rdf:Description rdf:about="links/link19548537L3">
      <lud:partOf rdf:resource="street/street19548537"/>
      <rdfs:label>Forest Ave</rdfs:label>
      <lud:lFrom rdf:resource="nodes/node202932241"/>
      <lud:lTo rdf:resource="nodes/node203060668"/>
      <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
      <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
      <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
   </rdf:Description>
   <rdf:Description rdf:about="links/link19548537L4">
      <lud:partOf rdf:resource="street/street19548537"/>
      <rdfs:label>Forest Ave</rdfs:label>
      <lud:lFrom rdf:resource="nodes/node203060668"/>
```

```
        <lud:lTo rdf:resource="nodes/node203046799"/>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
        <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
    </rdf:Description>
    <rdf:Description rdf:about="links/link19548537L5">
        <lud:partOf rdf:resource="street/street19548537"/>
        <rdfs:label>Forest Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node203046799"/>
        <lud:lTo rdf:resource="nodes/node202859448"/>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
        <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
    </rdf:Description>
    <rdf:Description rdf:about="links/link19548537L6">
        <lud:partOf rdf:resource="street/street19548537"/>
        <rdfs:label>Forest Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node202859448"/>
        <lud:lTo rdf:resource="nodes/node202832097"/>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
        <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
    </rdf:Description>
    <rdf:Description rdf:about="links/link19548537L7">
        <lud:partOf rdf:resource="street/street19548537"/>
        <rdfs:label>Forest Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node202832097"/>
        <lud:lTo rdf:resource="nodes/node203077627"/>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/TwoWayLink"/>
        <lud:length
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.1</lud:length>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node203009666">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9346517</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.960596</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node202832303">
```

```
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9361292</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9600914</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node202932241">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9375462</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9596343</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node203060668">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9389516</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9591784</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node203046799">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9401103</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9587876</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node202859448">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9413603</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9583447</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node202832097">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.9427657</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9579061</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
    <rdf:Description rdf:about="nodes/node203077627">
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">-
83.943785</geo:long>
        <geo:lat
rdf:datatype="http://www.w3.org/2001/XMLSchema#double">35.9575631</geo:lat>
        <rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Node"/>
    </rdf:Description>
```

To illustrate how a street is divided into links. The example of Dale Street from the example above is illustrated in the figure below.

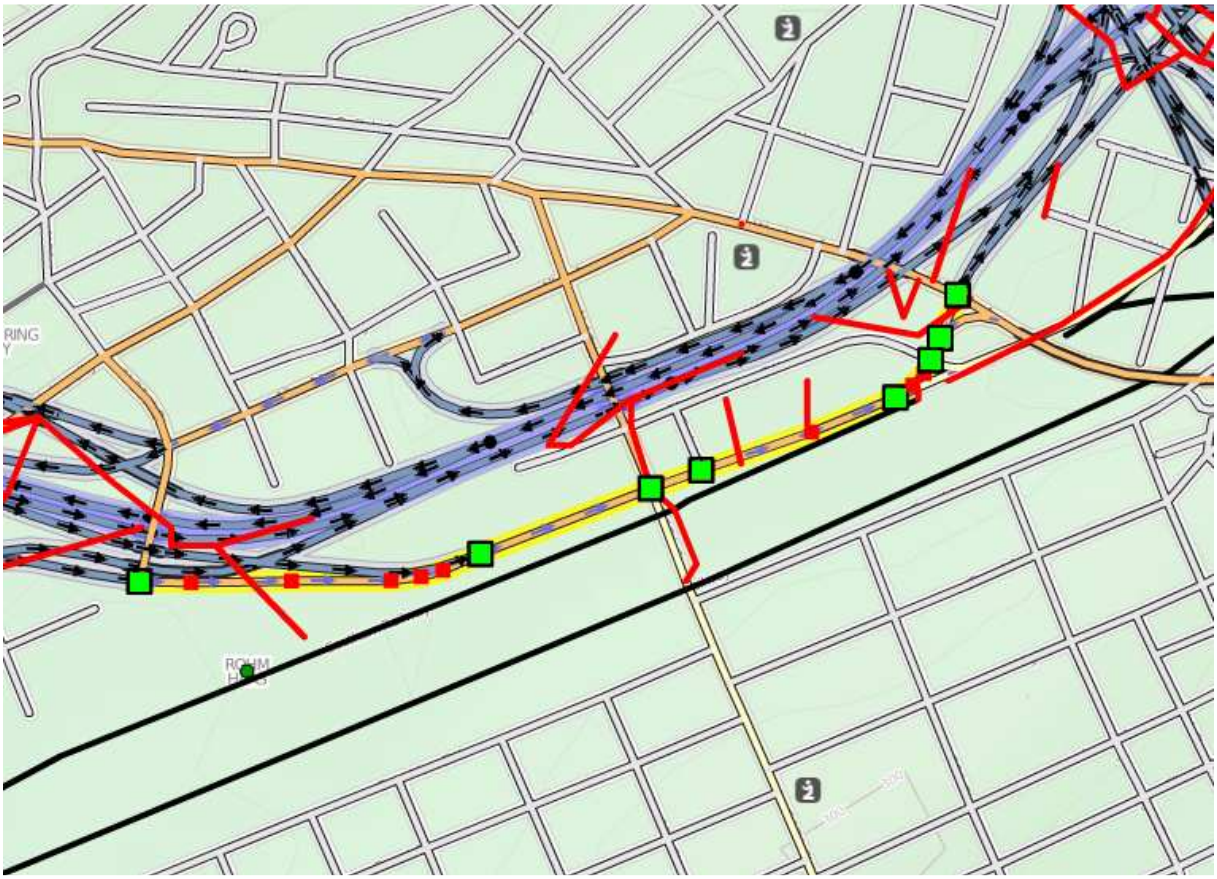

Figure 23: Dale Avenue broken up into links

In figure 23, the highlighted street is made up of sixteen nodes. Eight of these sixteen nodes are junction nodes because they are intersections of two or more streets. These junction nodes are the green ticks in the figure. The rest of the nodes are regular nodes and they are represented as red ticks.

According to the definition of links (A link is a relation that joins two consecutive junction nodes and all the regular nodes in between) hence this street should be divided into seven links.

15. Street ID = 19546326 Dale Ave
NODE 1 = 633052496
NODE 2 = 633053184
NODE 3 = 633053178
NODE 4 = 633053176
NODE 5 = 633053174
NODE 6 = 633053172
NODE 7 = 633052539
NODE 8 = 202908928
NODE 9 = 203058453
NODE 10 = 633064901
NODE 11 = 633064897
NODE 12 = 633064902
NODE 13 = 633064903
NODE 14 = 633064904
NODE 15 = 633064848
NODE 16 = 633064883

Figure 24: All the nodes of Dale Avenue

RDF	Comments
<rdf:Description about="links/link19546326L1">	Link 1
<lud:partOf rdf:resource="street/street19546326"/>	
<rdfs:label>Dale Ave</rdfs:label>	
<lud:lFrom rdf:resource="nodes/node633052496"/>	Junction Node
<lud:lEdge rdf:resource="nodes/node633053184"/>	Regular Node
<lud:lEdge rdf:resource="nodes/node633053178"/>	Regular Node
<lud:lEdge rdf:resource="nodes/node633053176"/>	Regular Node
<lud:lEdge rdf:resource="nodes/node633053174"/>	Regular Node
<lud:lEdge rdf:resource="nodes/node633053172"/>	Regular Node
<lud:lTo rdf:resource="nodes/node633052539"/>	Regular Node
<rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>	Junction Node
</rdf:Description>	
<rdf:Description about="links/link19546326L2 ">	Link 2
<lud:partOf rdf:resource="street/street19546326"/>	
<rdfs:label>Dale Ave</rdfs:label>	
<lud:lFrom rdf:resource="nodes/node633052539"/>	
<lud:lTo rdf:resource="nodes/node202908928"/>	
<rdf:type rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>	
</rdf:Description>	
<rdf:Description about="links/link19546326L3">	Link 3
<lud:partOf rdf:resource="street/street19546326"/>	
<rdfs:label>Dale Ave</rdfs:label>	
<lud:lFrom rdf:resource="nodes/node202908928"/>	
<lud:lTo rdf:resource="nodes/node203058453"/>	

```
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>
    </rdf:Description>
    <rdf:Description about="links/link19546326L4">                    Link 4
        <lud:partOf rdf:resource="street/street19546326"/>
        <rdfs:label>Dale Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node203058453"/>
        <lud:lEdge rdf:resource="nodes/node633064901"/>
        <lud:lTo rdf:resource="nodes/node633064897"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>
    </rdf:Description>
    <rdf:Description about="links/link19546326L5">                    Link 5
        <lud:partOf rdf:resource="street/street19546326"/>
        <rdfs:label>Dale Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node633064897"/>
        <lud:lEdge rdf:resource="nodes/node633064902"/>
        <lud:lEdge rdf:resource="nodes/node633064903"/>
        <lud:lTo rdf:resource="nodes/node633064904"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>
    </rdf:Description>
    <rdf:Description about="links/link19546326L6">                    Link 6
        <lud:partOf rdf:resource="street/street19546326"/>
        <rdfs:label>Dale Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node633064904"/>
        <lud:lTo rdf:resource="nodes/node633064848"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>
    </rdf:Description>
    <rdf:Description about="links/link19546326L7">                    Link 7
        <lud:partOf rdf:resource="street/street19546326"/>
        <rdfs:label>Dale Ave</rdfs:label>
        <lud:lFrom rdf:resource="nodes/node633064848"/>
        <lud:lTo rdf:resource="nodes/node633064883"/>
        <rdf:type
rdf:resource="http://www.linkingurbandata.org/onto/ama/Link"/>
    </rdf:Description>
```

Figure 25: Representation of the links in the RDF response

6. Conclusion and Future Developments

The outcome of the algorithm is very promising. The program retrieves accurately all the streets and nodes within the bounding box as well as other useful information and the RDF response contains all the information arranged as required in order to render it useful for other applications.

The execution time of the program is high due to large number of calls to the API as well as the large number of file read/write operation. API calls are time consuming since before each call a connection must be established and then closed after the call. The response time of API calls depends on the speed of the internet connection. A slow connection will slow down the response time of the API calls and will eventually affect the execution time of the algorithm. In theory, this application should run on a server hence the connection speed is usually better which will reduce the impact on the response time of the algorithm. File read/write operations are very time consuming and given that the number of these operations is greater or equal to the number of nodes within the bounding box therefore they will also have a big impact on the execution time. Other factors that have effects on the execution time are the density of streets and nodes in one the bounding box. Assuming the size of the bounding box remains the same, it will take more time to run the algorithm in area dense with streets and nodes then to run it in a less crowded area.

Since this is the first version of the algorithm, there are still many possibilities to improve it in the future.
The first improvement that comes to mind is modifying the way files are read and written. According to Sun MicroSystems the leading cause of poor I/O performance is failing to buffer I/O operations. Hard disks are very good at reading and writing sizable chunks of data, but they are much less efficient when working with small blocks of data. To maximize I/O performance, you should batch read and write operations [8]. Therefore, I/O operations such as writing OpenStreetMap xml responses to a file should be buffered. Another alternative could be reading the response and parsing it without writing it to a file which would eventually lead to a faster algorithm.
Depending on the final use of the application, if one is interested only in extracting links from the map therefore interested only in junction nodes then the algorithm could be modified to skip extracting regular nodes. By ignoring regular nodes, this means less web service calls to the API as well as file I/O operations thus leading to an improvement in the response time of the application.

Another possible development/modification could be using the function bbox defined in OpenStreetMap.
The following command GET /api/0.6/map?bbox=*left*,*bottom*,*right*,*top* returns:

- All nodes that are inside a given bounding box and any relations that reference them.

- All ways that reference at least one node that is inside a given bounding box, any relations that reference them [the ways], and any nodes outside the bounding box that the ways may reference.
- All relations that reference one of the nodes or ways included due to the above rules. (Does not apply recursively.)[1]

Yet another development for this application is to expose it as a web service. Since the application is meant to reside on the server side, a web service interface would allow it to be accessible via HTTP. So, the arguments (Initial node and the dimension of the bounding box) could be sent to the application by a remote user through an HTTP Get request and they would get the result returned as an xml/RDF response.

7. References

[1] Open Street Map, Wiki Main Page, Available online at http://wiki.openstreetmap.org/wiki/Main_Page.

[2] Open Street Map, Wiki Main Page, Available online at http://wiki.openstreetmap.org/wiki/Elements

[3] Manola, F. and Miller, E., 'RDF Primer', World Wide Web Consortium, Recommendation, 2004.
Available online at http://www.w3.org/TR/rdf-primer/

[4] Open Street Map, Wiki Main Page, Available online at http://wiki.openstreetmap.org/wiki/Planet.osm

[5] Richardson, Leonard; Ruby, Sam (2007-05), RESTful Web Services, O'Reilly

[6] G. Klyne and J. Carroll. W3C recommendation, RDFconcepts and abstract syntax.
http://www.w3.org/TR/rdf-concepts/, February 2004.

[7] Open Street Map, Wiki Main Page, Available online at http://wiki.openstreetmap.org/wiki/Beginners%27_Guide

[8] Steve Wilson and Jeff Kesselman, Java Platform Performance Strategies and Tactics, Sun Microsystems 2000. Available online at http://java.sun.com/docs/books/performance/1st_edition/html/JPTOC.fm.html

[9] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2):199-220, 1993. Available online at http://tomgruber.org/writing/ontolingua-kaj-1993.htm

[10] Emanuele Della Valle, Irene Celino and Daniele Dell'Aglio. LarKC Project. Available online at http://wiki.larkc.eu/LarkcProject/WP6/WorkInProgress/AMAData