Automation of process to load database from OSM for the design of public routes

G. Bejarano, J. Astuvilca, P. Vega
Pattern Recognition and Applied Artificial Intelligence Group
Pontifical Catholic University of Peru
1801 Univesitaria Avenue, Lima, Peru
gissella.bejarano@pucp.edu.pe, {j.astuvilcaf, vega.pedro}@pucp.pe

Abstract

This paper presents an automatic process to load the transport information of Lima - Peru city as network and public transportation routes from *OpenStreetMap* (OSM) to a *PostgreSQL* database. Moreover, this work shows how OSM data is transformed in two graphs and in several actual bus routes. The work is a combination of SQL commands and python programs to transform OSM data and combine it with external information to specific structures in a final database. This information was necessary for a second goal that was the approach and study of the Transit Network Design Problem (TNDP). Since OSM is a free collaborative tool, it is subject to manual errors in the map that produce mistaken graphs and routes. These errors are corrected daily because the area information is updated frequently. The obtained results confirm that this process could be automated in a one-click step. Finally, we tried other methods to upload OSM data and none of them got an exact graph except for *osm2pgrouting*.

1 Introduction

Until 2014, the city of Lima, capital of Peru, had 403[1] formal routes, 1 Bus Rapid Transit[2] (BRT) line and 1 metro line. There are projects to build a new BRT corridor[3] and 5 more metro lines[4], in order to improve public transportation. Actually, some months ago, the number of formal bus routes was reduced to 322 urban routes, 77 beltway routes and 15 routes in unattended zones[5] because of the oversupply of routes that exist in the city.

The task of deciding which set of bus routes must continue in order to attend the public mobility demand and to optimize the cost of the user and the cost of the operator is a huge and full of possibilities task (Farahani et al., 2013). For this reason, a metaheuristic algorithm might be used to find a nearly global optimal solution (Farahani et al., 2013) to this problem defined by Fan and Machemehl (2004) as the Transit Network Design Problem (TNDP). Besides, according to Fan and Machemehl (2004), metaheuristics are more suitable to work with practical and real cases. Nevertheless, a real problem was to load a complete network as an input to the algorithm and form a graph by which a set of routes will pass. For this reason, the purpose of this work is to implement an automatic process that provides the necessary data input to an algorithm that solves the TNDP.

For our algorithm, two levels of graphs would be tested in order to find a solution progressively. The first graph represents the adjacent Traffic Zones (Agency, 2005), which we call minizones because they can be smaller boundaries in the future, while the second graph is the road and highways network level. First, sequences of minizones are created to generate general routes that satisfy soft and hard restrictions. Then, using these first routes, real bus routes are defined in a network of roads level. Once we have all the necessary information for the algorithm, the sequence and calls of execution are organized in a single executable command.

[1] http://lima.datosabiertos.pe/home/

[2] https://www.itdp.org/library/standards-and-guides/the-bus-rapid-transit-standard/what-is-brt/

[3] http://archivo.larepublica.pe/11-04-2015/municipalidad-de-lima-castaneda-anuncia-ampliacion-de-la-ruta-lima-norte-del-metropolitano

[4] http://www.aate.gob.pe/metro-de-lima/

[5] http://www.elperuano.com.pe/NormasElPeruano/2015/02/28/1205528-1.html

This work is organized as follows. Section 2 explains the information sources for the database created to keep the data needed and the solutions produced by the algorithm for the TNDP. Section 3 describes the extraction and transformation processes used to get the information mentioned and load it to a default database. Section 4 explains the organization of the files and instructions used in the single process and the order in which they are executed to automatically finish the process. Conclusions of this work are presented in Section 5 along with future recommendations.

2 Sources of Information

To solve a basic case of the TNDP, two sets of data are needed as we can see in Mautone and Uqhuart's work (2009). The first group is the network that generates one of the graphs required to evaluate solutions at a network road level, and information about the edges of this graph like the road owner of those edges and the road's type. The second group consists of the demand information, the number of travels made from an origin zone to a destiny zone and the adjacency graph among zones. Using the Pair Insertion algorithm (PIA), random routes that cover certain percentage of the total demand of the zones are generated. Besides those two sets of groups, Lima has an actual real solution and that is why we decided to test this one as an initial solution. The actual information about routes would be combined using a genetic algorithm in different forms in order to optimize the set of routes that solve the TNDP in both levels of graphs: zones and network.

The next two subsections describe the sources for all the groups of data mentioned.

2.1 Road Network

The two sources analyzed to generate the public transportation network are shown in Figure 1. The first source was the one given by the Ministry of Transportation and Communications of Peru. The problem with this source was that it was difficult to identify the edges and nodes of the network due to the format of the file: shapefile[6] (.shp), i.e. one set of edges was represented as a single edge and vice versa. In order to verify the correctness of the shapefile, it was loaded into OSM and in some cases, the roads were displayed crossing a block. The second analyzed source was OSM, which has user generated geographic content (Janssen and Crompvoets, 2012). After some transformation, OSM produces a valid, but not official, graph that represents the public transportation network. This network would be enough for our TNDP studies. Besides, this contribution on OSM could be used for future studies.

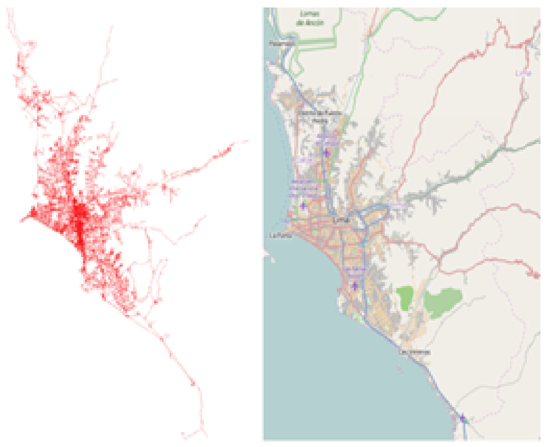


Figure 1: Difference among MTC Network (left) and Network from OSM (right)

2.2 Routes

In order to get the actual public transportation routes, two complementary sources were analyzed. The first source was the Metropolitan Lima Municipality, which has one SHP file with the whole set of routes. A process to get a SHP file per route was made through the *ogr2ogr*[7] command and a batch program. After the command produces a comma separated values file (.csv) which contains a *WKT*[8] (or *LineString*[9]) for each route, the batch program creates a SHP file from every record of the CSV file. Once a route is individually identified, it is drawn manually in the OSM interface and matched with all the ways (set of edges) by which it passes in order to maintain the geometric consistency with the road network. In this process, a lot of errors are generated and they would be solved thanks to daily reports that indicate their coordinates and details.

[6]Geospatial vector data format for Geographic Information System (GIS) Software.

[7]A command line utility from the "Geospatial Data Abstraction Library" (GDAL).

[8]Well-known text, markup language for representing vector geometry objects.

[9]Vector geometry object that represent a line.

2.3 Demand

Due to a study of demand and transportation in Lima made by the Japan International Cooperation Agency (2005), a set of 427 Traffic zones was established in the city of Lima. This information implied two types of data: polygons that represent the zones and the number of travels made among every pair of O-D zones (origin and destiny). The polygons are used to delimit a set of edges for which a route must pass by in order to satisfy the demand of that zone (like an origin or destiny).

3 Transformation process

This section describes the steps followed to extract data from OSM sources, upload it into a database and transform it into a final database called routes.

3.1 Extraction from the sources

In order to get the OSM data about the public transportation network and routes of Lima, the files containing the information of Peru and the boundary of Lima must be downloaded and used in a command line. There is a server named *Geofabrik*[10] which has data extracts of countries of all the continents and is updated daily (Zielstra and Zipf, 2010). However, as this server provides the information of the whole country, a boundary must be applied to get the information of only the city of Lima. This can be done through an OSM relation ID which can be obtained from the *MapIt*[11] website, through the insertion of a coordinate (latitude, longitude) of the objective city. After that, the OSM relation ID is introduced in a polygons generator page[12] and the .poly file is downloaded.

Osmosis is the command line application from OSM with which a file containing the information of the primary, secondary and tertiary highways of Lima is produced as well as the trunk, motorway, residential and their links. Figure 2 shows how the osmosis command takes as input the information of Peru (peru-latest.osm.pbf) and the boundary of Lima (lima-callao.poly) and produces the lima-callao.osm file.

3.2 Default and final databases

Once the lima-callao.osm file is produced, it is necessary to upload this information to a database to manage the information. For this task, a tool

[10]http://download.geofabrik.de/

[11]http://global.mapit.mysociety.org/

[12]http://polygons.openstreetmap.fr/

```
osmosis --rb file=peru-latest.osm.pbf
--bounding-polygon file=lima-callao.poly
--way-key-value keyValueList="highway.primary,
highway.secondary, highway.tertiary, highway.trunk,
highway.motorway, highway.residential,
highway.motorway_link, highway.trunk_link,
highway.primary_link, highway.secondary_link,
highway.tertiary_link" --write-xml lima-callao.osm
```

Figure 2: Osmosis command to get .osm file

like *osm2pgrouting*[13] was used. It provides a process that converts OSM data into a topology and it is uploaded in database. First, we must create a database called *pgrouting-workshop*[14]. After that, the command shown in Figure 3 must be executed.

```
osm2pgrouting -file lima-callao.osm
-conf mapconfig.xml -dbname pgrouting-workshop
-user postgres -clean
```

Figure 3: Osm2pgrouting command to create pgrouting-workshop database

This command filters the road types mentioned in mapconfig.xml and create tables like *ways* (edges), *classes* (types of roads), *nodes*, *relations* (routes for example), *relation_ways* (which relates the edges of a route), *types*, *way_tag*, *way_vertices_pgr*. The full schema for this database is shown in Figure 4.

Certainly, there are other tools to import OSM data into a local database (*osm2pgsql*, *imposm* and *osmosis*). However, only *osm2pgrouting* builds an exact graph with edges and nodes, and Section 3 will show how that makes a difference.

In order to have the direct information of the sources and the information of the application or algorithm separated, a new database *routes* is created. Table 1 shows the source of every table in the new database (from default *pgrouting-workshop* database or from external data).

3.3 Transformation

In this section, a brief logic of the load of every table would be shown. See the Figure 5 for more details. every table would be shown.

The table *road_types* contains the different types of roads that are presented in Lima's OSM data

[13]http://pgrouting.org/docs/tools/osm2pgrouting.html

[14]ftp://ftp.remotesensing.org/pgrouting/foss4g2010/workshop/docs/html/chapters/topology.html

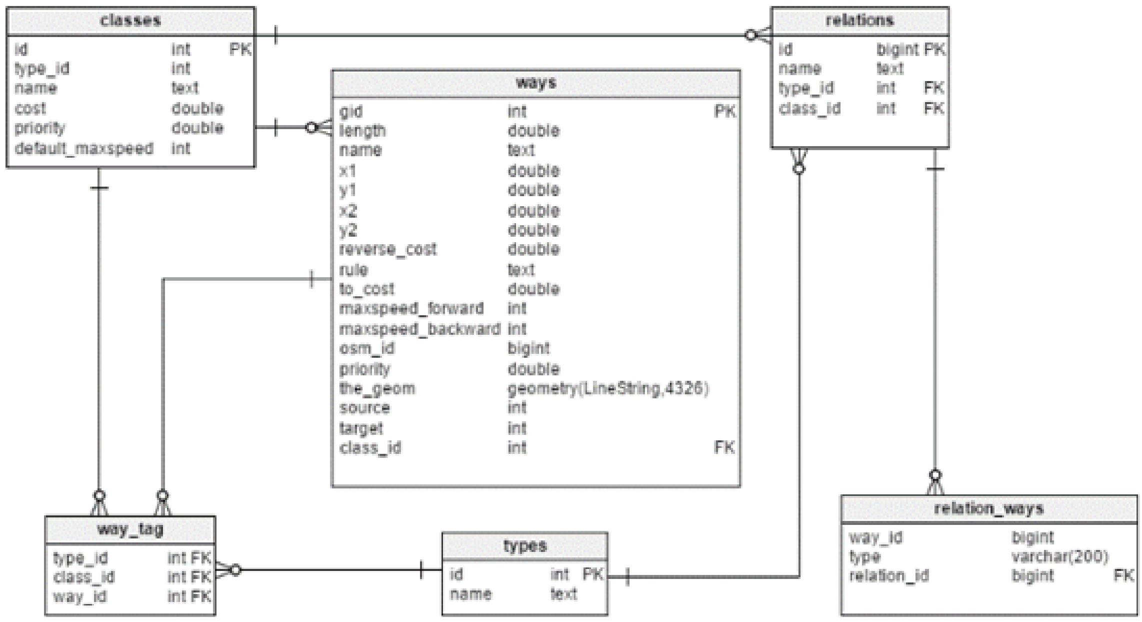


Figure 4: Pgrouting-workshop schema database

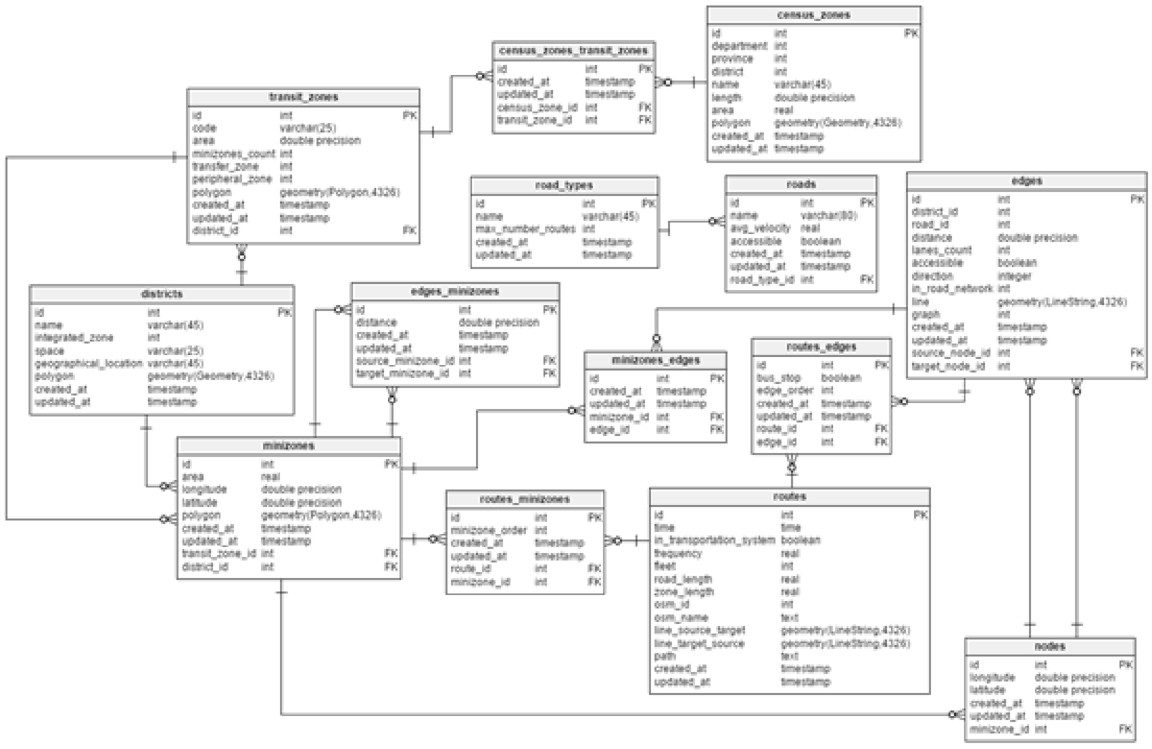


Figure 5: Routes schema database

Entity	Database	External File
RoadType	Classes (default)	
Road	Ways (default)	
Node	Ways (default)	
Edge	Ways (defaut)	
District		i4_districts.csv
Minizone		i8_census_zones.csv
Demand		demand_matrix.csv
Route		lima-callao.osm
RouteEdge	Ways (default)	list_final_routes.csv
RouteMinizone	routes, routes_edges, edges, transit_zones (routes)	

Table 1: Input and output of tables from routes database.

like primary, secondary and tertiary road among others. Besides, this table has an additional field (not from OSM) that indicates the maximum number of routes that would be used in the future when solving the TNDP.

The table *roads* is filled from the table ways of the *pgrouting-workshop* database, which contains the longitude and latitude of both nodes that form an edge (known as source and target). However, as a road is formed by several edges or ways, a distinct query by the name of the way is made to obtain unique roads.

The table *nodes* is filled by searching every way and saving each source or target as a unique key in a hash table. Besides that, a function from the *PostGIS* extension is applied to define which minizone every node belongs to. Additionaly, the *edges* table is similar to the table ways (on *pgrouting-workshop* database) but the *road_id* is brought from the previous step (*roads* table) to complete its load.

We use several steps to fill table *routes*. First, a file is generated from the lima-callao.osm file containing just the relations-routes from the users of the project. After that, just the routes which do not have any errors are listed in a CSV file and they are finally uploaded to the table.

The process to define which edges belong to certain route has been a continual feedback. First, a hash table of different sources and targets nodes from the table *ways* was created. Second, every node was linked to its respective previous and next node. Each node has its own *edge's gid*[15], relative to the current edge. After that, a search is made to identify which nodes (source or target) are used in the graph.

[15]Road link if of the table ways.

Finally, a whole loop is made to search along the hash table from the start node and get the edge's to which the actual node belongs to and its respective order. It is important to mention that there are some mistakes in this logic due to some errors or missing information in the direction of the ways (edges).

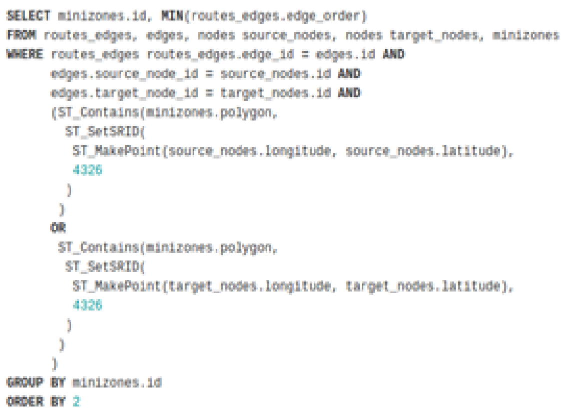
Based on the filling of the route's edges, the logic to fill the table *routes_minizones* is to analyze the source and target node of every edge that form the route. As every node belongs to a minizone, a list of every minizone that contains a route's nodes is made. Moreover, this list is ordered by the edge's order calculated in the previous step as an attribute of the table *routes_edges*. This list is grouped by the *minizone_id* to avoid the repetition of a minizone on different edges. This logic is applied using some functions of the *PostGIS* extension like *ST_Contains(polygon, point)* that decides whether a point is contained in a polygon or not and *ST_SetSRID(point, system)* that sets the 4326 system reference[16] of a point. This logic is better shown in Figure 6.

When nodes are on the limit of the polygon like the boundary of a demand zone, they could belong to more than one zone. However, this work did not analyzed which minizone is selected by the function *ST_Contains* from the *PostGIS* extension.

4 Results: Automatic process organization

Before getting a stable database in which you can execute an algorithm to the TNDP, several databases loads must be done in order to evaluate the accuracy of the routes drawn manually. That

[16]`http://suite.opengeo.org/opengeo-docs/glossary.html`



```
SELECT minizones.id, MIN(routes_edges.edge_order)
FROM routes_edges, edges, nodes source_nodes, nodes target_nodes, minizones
WHERE routes_edges routes_edges.edge_id = edges.id AND
      edges.source_node_id = source_nodes.id AND
      edges.target_node_id = target_nodes.id AND
      (ST_Contains(minizones.polygon,
        ST_SetSRID(
         ST_MakePoint(source_nodes.longitude, source_nodes.latitude),
         4326
        )
       )
      OR
       ST_Contains(minizones.polygon,
        ST_SetSRID(
         ST_MakePoint(target_nodes.longitude, target_nodes.latitude),
         4326
        )
       )
      )
GROUP BY minizones.id
ORDER BY 2
```

Figure 6: SQL script to fill table minizones

is the reason why an automatic process was set to run daily. In Figure 7, a sequence of programs, commands, input and output files are shown to explain the process of downloading the information from OSM, combine it with external information and load them to a final database.

```
python DownloadAndBoundary.py
./2_pgrouting/osm2pgrouting.sh
python TransformEntities.py
python 3_scripts/python/RouteReports.py
python UploadFinalDB.py
```

Figure 7: Content of executable final.sh

Some tools must be installed in the server and the local computer before executing this process. These are: *gdal*, *osmosis*, *PostGIS*, *pgrouting*[17] and *psycopg2*.

The automated process has a series of steps called from an executable file (final.sh) in Linux. It lasts 30 minutes approximately and the topology (graph) size is about 150000 edges and 100000 nodes; the number of routes is 300. The content and the structure of the commands and process called from final.sh are shown in Figure 8. Also, the process generates error reports about the current drawn routes, for each one evaluates how many edges exist in each node, so if more than two edges exist in one node, then it reports that the route has an error. This process was carried out daily until no error is found in the routes. It is important to mention that this process is recommendable when working in a local database where the password could be stored in files to allow the interaction of calculus inside and outside the database.

[17]`http://pgrouting.org/`

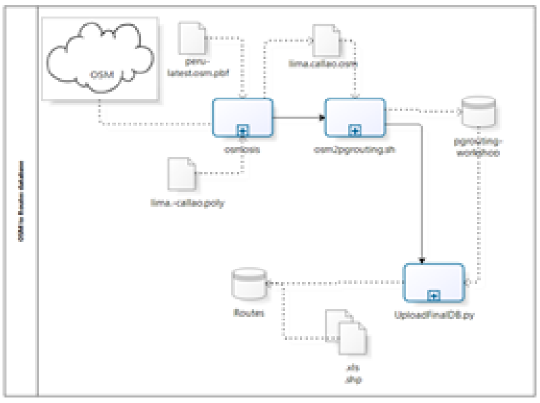


Figure 8: Diagram of process of downloading data, transforming it and uploading it to the database (Tool: Bizagi)

5 Conclusions and recommended work

This section presents the conclusions after implementing an automated process for uploading OSM data combined with external information. One of them was the confirmation of a tool that generates the topology of the map or graph rather than other tools that also upload the same data but in a different scheme.

5.1 Conclusions

In subsection 3.1, it was mentioned that there were other tools to upload OSM data. *Osm2pgsql* and *imposm*, after installed and executed, generate a table where the information of ways can be found. However, the field that represents the geometry does not generate the sequences of edges. Actually, *osm2pgsql* generates edges that are not necessary for the graph and make impossible to distinguish which ones are. We could have worked with edges that were not required but it would have been an unnecessary addition of data to a problem that is already complex. In addition to that, *imposm* generates the same edges composed of only the first and last node of the way. The manner these edges are stored in these schemes (*osm2pgsql* and *imposm*) hinders the recognition of edges in a way as seen in Figure 9 where just three edges (*osm2pgrouting*) should be generated from the selected way instead one (*imposm*) or seven (*osm2pgsql*).

On the other hand, there are a lot of routing applications that use OSM data to combine it with other type of information at some point (Amat et al., 2014)(Vetter, 2010). However, some of them

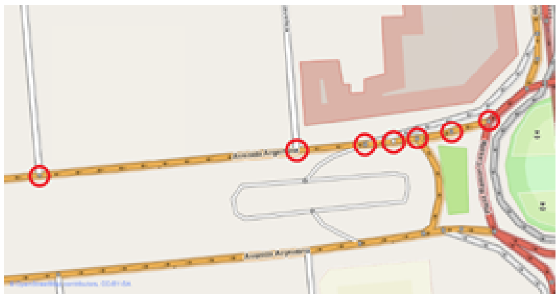


Figure 9: Shows the numbers of nodes counted in one way: Argentina avenue - Lima, Peru (Obtained from OSM and modified) (Tool: Bizagi)

are implemented just for certain cities, for other routing problems like private cars and bicycles or just do not work very well[18]. We found some commercial routing applications but obviously they do not public the process to combine their sources into a unique database.

5.2 Recommended work

As OSM is a free collaborative tool, more analysis is recommended in order to establish several logics that allow us to maintain the coherence of the data despite of new errors.

A full review of the possible scheme obtained and filled from *osmosis* should be finished in order to know if there is a faster and simpler way of loading the information. However, it seems that there is no direct form to identify the relations routes with the osmosis' scheme. This is because of the different order that the *route* tag has in a field to recognized that a relation is a public transportation route.

There is also another tool that seems to convert OSM data into a graph topology that must be analyzed: *OSM2PostGIS*[19]. However, it is still in an early development.

References

Japanese International Cooperation Agency. 2005. Plan maestro de transporte urbao para el área metropolitana de lima y callao en la república del perú fase 1–9. problemas y temas actuales del transporte urbano.

Guillermo Amat, Javier Fernandez, Álvaro Arranz, and Angel Ramos. 2014. Using open street maps data and tools for indoor mapping in a smart city scenario.

Wei Fan and Randy B Machemehl. 2004. Optimal transit route network design problem: Algorithms, implementations, and numerical results. Technical report.

Reza Zanjirani Farahani, Elnaz Miandoabchi, WY Szeto, and Hannaneh Rashidi. 2013. A review of urban transportation network design problems. *European Journal of Operational Research*, 229(2):281–302.

Katleen Janssen and Joep Crompvoets. 2012. *Geographic Data and the Law: Defining New Challenges*. Leuven University Press.

Antonio Mauttone and María E Urquhart. 2009. A route set construction algorithm for the transit network design problem. *Computers & Operations Research*, 36(8):2440–2449.

Christian Vetter. 2010. Fast and exact mobile navigation with openstreetmap data. *Master's thesis, Karlsruhe Institute of Technology*.

Dennis Zielstra and Alexander Zipf. 2010. A comparative study of proprietary geodata and volunteered geographic information for germany. In *13th AGILE international conference on geographic information science*, volume 2010.

[18]`http://wiki.openstreetmap.org/wiki/Routing/online_routers#Route_service_comparison_matrix`

[19]`http://pgrouting.org/docs/tools/osm2PostGIS.html`