CoopMap: Democratizing Community Maps using Crowdsourcing

Ricardo Xavier
ricardojosexavier@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2019

Abstract

Nowadays, we depend on information more than ever, and maps are one of many other forms in which we can represent that information. As users, we can find many map-based applications that provide specific information about public spaces. However, we may not find every detail we were looking for in those applications.

The idea of this thesis is to develop a crowdsourced and mobile platform where users can create a map and classify streets according to attributes that are relevant to a given community. This way, any map can provide any type of information, as long as there is a group of people interested in it.

CoopMap is a platform designed to fulfill this gap of information between communities and their needs. It was developed for Android devices and uses OpenStreetMap to extract data about the streets. Within a few steps, this platform allows users to create their map with a set of attributes. They also can join other users’ maps. When inside a map’s instance, users can contribute by submitting spatial judgments about streets or they can just visualize the results from other users’ contributions.

For this thesis, we have produced a prototype that was evaluated. We wanted to know if the proposed solution could answer to the necessities of the user. We have performed some benchmarks and a usability test. From the results obtained, we were able to conclude that our platform meets the requirements. However, this does not mean it cannot be improved for future work.

Keywords: Crowdsourcing, Volunteered Geographic Information, OpenStreetMap, PostgreSQL

1. Introduction

A driver usually wants to know which is the fastest way to get somewhere and there is a lot of applications that assist users in that matter like Waze [12]. However, people are requiring other sorts of information which are not that accessible. An example is the disabled people that barely have any information about the streets they are going to pass through with their wheelchairs. Imagine the case where a disabled person must reach a certain address using his/her wheelchair and the sidewalk is in a bad state that might result in a trapped or broken wheelchair. What if there was a way of reporting this street’s condition to other people in the same physical state so that they can avoid passing by this specific street? Mobasheri et al. [18] created a solution for this concrete scenario. However, this solution only targets one problem among many others. A typical person that goes for jogging enjoys running in a track with the pavement in a decent state (to avoid getting injured), not too crowded (to not disturb or get disturbed by other citizens) and probably with a good air quality index (low pollution levels). Many other aspects promote a good jogging activity, however, we only described three to demonstrate how complex a spatial assessment can be. A map gathering all these sorts of information would be appreciated by the jogging community.

Some solutions innovate by using crowdsourced map information to solve particular problems ([18], [15]). These types of solutions aimed at solving one single issue and are limited to the developer’s perspective of the problem. Some platforms offer the user the possibility of creating custom maps with any type of information (Google MyMaps and Wikimapia[14]). However, they do not provide proper mechanisms to collect, in a crowdsourced fashion, georeferenced data that complement the base information of the underlying map.

Building one such application from scratch requires a significant amount of knowledge that most of the users do not own. A platform accessible for any non-programmer user that allows the creation of custom maps would lead to a variety of maps with every kind of information. This means that
there would be maps today not available anywhere else, and in which the user can rely on when he needs it.

This thesis aims at developing a crowdsourced mobile application where users can create and have access to any kind of map without requiring any special skill. These maps have a set of attributes defined at creation by the user. When the map set-up is completed, the user can start picking streets or just segments (small parts of a street). After these segments are selected, he classifies these areas regarding the attributes defined during the map creation. Hereafter, we call each classification provided by a user on a given map segment as a *contribution*. Other users can join the map and submit contributions as well. The application provides a way of visualizing the results from these contributions. Consequently, the communities can visualize, interpret and use that specific map information for personal use.

We decided to create a platform using a client-server approach. The client consists of a mobile application developed to the Android operating system, which allows our users to interact with maps in two different ways. In **Visualization Mode**, they can visualize an instance of a map (belonging to a specific community) and obtain information about other users’ contributions. In **Contribution Mode**, the user can select streets (or parts) and submit contributions (considering that map’s attributes).

The back-end server was implemented using the gRPC technology and it is mainly responsible for processing the multiple contributions coming from users.

We use two distinct databases near the server. OpenStreetMap copy database is responsible for maintaining the metadata from the streets which comes from OpenStreetMap [7] platform. This data is used used to populate or update the second database. The second database is called the CoopMap database and contains the data to be accessed in run-time by our users.

After implementing our solution, we performed a formal evaluation of our platform. During this evaluation, we made multiple benchmarks and a small usability test.

2. Related Work

2.1. Crowdsourcing

Crowdsourcing is the concept of outsourcing a piece of work so the crowd can perform it in exchange for some kind of payment or benefit. The crowd in this context are the users of the platform that are looking for work to perform or also called the *workers*. The people interested in the work performed by the crowd are called *requesters*. The work assigned to a certain *worker* is called a *task* and the results delivered by the crowd are called *outputs*.

In the context of this thesis, a user can play the role of *requester* and *worker* at the same time. Thus a user can provide or receive information for personal use. The role of the *requester*, in this case, is a user that is interested in specific information but does not assign *tasks* directly to *workers*. Instead, *workers* should have the initiative of voluntarily choosing the *tasks* they wish to contribute to.

2.1.1 Traditional Crowdsourcing vs Participatory Sensing

Participatory sensing uses embedded sensing (e.g. mobile phones) and communication (cellular or WiFi) infrastructure to collect data ([22] [16]), resulting in a reduction of the user’s involvement. When a user is moving, it is possible to get information about his speed through the accelerometer of his smartphone or other wearable mobile device and send that information to a running server to process it. This is a brief example of a sensing activity. Traditional crowdsourcing, on the other hand, implies users manually performing tasks. This second way of crowdsourcing is more suitable for tasks in which humans perform way better and faster than computers while participatory sensing is more appropriate for repetitive and straightforward tasks.

Participatory sensing depends mainly on mobile phone’s sensors as described previously and therefore there is a major concern we need to have in the account. The information we can get from sensing is very limited to the kind of sensors we have. For instance, if a certain community is looking for information about good areas to walk their pets, it is very difficult information to acquire from sensors. However, there is ongoing research on how to optimize the use of the smartphone’s sensors. For instance, NoiseTube [17] is an application where we can measure noise pollution levels using the microphone.

Traditional crowdsourcing requires human effort, which might be a big drawback since individuals need to be motivated to perform some sort of tasks [21]. But since it calls for user attention, the benefit the user obtains from performing the task must correspond to the cost of performing it. This is, in many cases, the biggest disadvantage. Besides this, there is another disadvantage we feel the need to refer, which is: traditional crowdsourcing is more susceptible to human mistakes and as a result requires detailed and better-defined quality control measures to avoid using inaccurate data. The main advantage of this type of crowdsourcing...
is that we can get any type of information as long as anyone is willing to provide it.

Participatory sensing does not mean that a user will not interact with the application once in a while. However, the goal of a system that applies this approach is to absorb most of the information through sensors. Our goal is to make users manually gather most information. Some sensors may be used to help the user with his activity. However, their outcome is not meant to be shared with other users. Because of these two points, we classify our approach as traditional crowdsourcing.

### 2.1.2 Volunteered and Rewarding models

In the **volunteered model**, the requesters depend on volunteered work performed by the crowd. For a volunteered crowdsourcing platform to have success, it is required a strong sense of community between the requesters and workers. To achieve this, usually, there is a communication channel where these two groups can have contact with each other and share their ideas (for example an internet forum). This way, the crowd may be aware of the requesters’ goals inside the platform and may decide help. Workers must know how their contributions will affect the course of the project.

The **marketplace model** is one of the most popular approaches in the topic of crowdsourcing. Some of the most relevant platforms like Amazon Mechanical Turk [1] and Microworkers [6] apply this model. The analogy in the name makes the concept very self-explanatory.

When comparing both models, it is noticeable a behaviour difference on workers [19]. The level of commitment in the marketplace model is bigger than in the volunteered model. The reason behind this is because in a volunteered platform, usually, a worker does not lose much when he quits from a task before its completion. On the other hand, in a marketplace model, when a worker does this, usually, it means he is not getting paid. It is also possible, as expected, to find more workers performing tasks while getting paid than when volunteered. However, there is one aspect where the paid model loses to the volunteered one which is the reliability of the worker’s answers. Redi et Povoa [19] conducted a study which proved the previous statement. The cause associated with this fact is related to the worker’s motivation when performing the task. In a paid model, the worker is interested in getting paid no matter what is the result of his work, while in a volunteered model, the worker is interested in his contribution’s impact on the project and therefore tends to perform better.

Regarding this document's topic, it is more common to see solutions using a volunteered model because tasks are usually simpler and faster to perform. There is also another important factor regarding this project which is: users benefit from other users’ contributions which means there will be a stronger sense of community. The user motivation increases when knowing that a task’s outcome will affect multiple users.

### 2.2. Volunteered Geographic Information

The concept of users mapping geographic information on dedicated platforms is called **Volunteered Geographic Information** (VGI). Users can be trained but most of the time they are not. There are two types of Volunteered Geographic Information ([20]).

**Explicit-VGI** is the process of a user volunteering to provide information for a project while knowing its purpose. Some platforms using this approach are OpenStreetMap [7], Wikimapia and Google MyMaps.

In **Implicit-VGI**, the users provide information which they do not know for which purposes it will be used. An example is when applications extract geographic information from Twitter [11] posts (tweets) like the location of a music concert. Platforms like Wikipedia [13] and Flickr [2] use this way of gathering information.

For this project, it is intended that users contribute with information manually and directly, knowing it will be used to improve maps for communities. In other words, an explicit approach is more appropriate.

### 2.3. OpenStreetMap [7]

OpenStreetMap is a VGI platform where a user can create an editable map of the world for free (like Wikimapia and Google MyMaps). It was meant to overcome some of the restrictions related to the use and availability of map information. Its data comes mostly from the users via the manual survey, GPS devices, aerial photography and other free sources. This data is then stored in a shared database which can be updated.

OpenStreetMap has one main database which is shared by everyone. However, if a user wants a private map instance to edit, it is created an instance of another database specific for that user. The main copy of OSM is stored in the main database which is an instance of an open-source object-relational database called PostgreSQL [10]. When someone edits something, the main database is altered but the map only gets updated a few minutes later to avoid rendering whenever a user changes something.

OpenStreet map has four data primitives which are:

- **Node**: a pair of coordinates that specifies a
geographic position according to the standard World Geodetic System (WGS84). It is used to mark points of interest, mountain peaks and also to create ways.

- **Way**: ordered list of nodes which represent a connected sequence of line segments. They can represent streets, rivers and other linear elements of a map. They can also represent areas (e.g. lakes or parks) if they form a closed loop.

- **Relation**: ordered list of nodes, ways and relations, where each element can have a role (string). A relation is used to represent a connection or relationship between nodes and ways. An example would be a relation between a way that represents a road and a node that represents a stop sign in that road.

- **Tag**: key-value pair that is used to store metadata about a map object (e.g. name, properties or type). It is always associated to an object (node, way or relation).

Each data primitive has its table and individual objects are stored as rows in these tables. Every single edit happens in this database. The user can keep his customized map by downloading a planet.osm file. This file is composed of all primitive data instances and its size is over 160GB. It is possible to keep a small part of the map using a tool like Osmosis Java Software [9].

Besides Osmosis, OSM also recommends and provides other tools to help developers designing their applications. The most relevant we found were osm2pgsql (takes an OSM XML file and loads it into a database) and mapsforge [5](provides tools and APIs for rendering, overlays and downloading maps).

**3. Implementation**

CoopMap consists of a mobile application where users can either create a map or join an existing one. After this initial step, they can start contributing to this instance of a map.

A contribution, in the context of this platform, consists of a subjective evaluation of the attributes defined considering some specific street or area. Attributes are spatial characteristics defined by the map creator. Consequently, this evaluation (or contribution) is associated with only one map’s instance. During the process of contributing, for each attribute, the user has to give a score with the minimum value of one and a maximum value of five. These values were inspired by how users can evaluate mobile applications on the Google Play Store. When a user submits a contribution, the value decided by the user for each attribute is merged with other users’ values. This will induce a variation in the original value. It is important to refer that each user can only submit one contribution for each street segment. The score of a street segment follows an Arithmetic mean taking into consideration all the attributes of the map.

In a map, the streets’ overall score is fully decided based on users’ contributions. This overall score is calculated considering an average between all attributes defined by the map owner. Because the city is always changing, it was decided that we should give more weight to contributions made in a more recent period. To obtain a value for a certain interval of time we calculate the average of the contributions submitted (by any user) in that interval.

We wanted to make a recent contribution more valuable than an older one without making the older one total meaningless. Also, we did not want to have a binary approach where a contribution is either old or recent. Considering these two conditions, we decided to implement a system where we attribute weights to different time intervals. All contributions are distributed between all these time intervals taking into consideration the moment these contributions were submitted. Newer contributions will have higher weights while older ones will have lower weights.

To better understand our solution, consider a time scale from the first day of one year until the last day of the same year. Consider as well that we are currently on this last day of the year and that each time interval coincides with each month of the year. For a certain street segment A, if there were twelve contributions submitted, one for each month, the result would be twelve contributions, all with different impacts on the overall score of the street segment. The first contribution, which was made in the first month of the year, will have a lower impact than all the following contributions made in the next months. The last contribution (made in the last month of the year) will have a higher impact than all the previous ones. While the time passes, contributions will shift to the next interval of time and will have a lower impact on a street’s segment overall score.

In the long term period, some works can alter some of the attributes of a street. For instance, if one of the attributes of a map instance is the state of the sidewalk, that can easily change with some street works. Distributing weights between these intervals, in the long term will achieve better results.

**3.1. Back-End Server**

To build the server, we used a framework called gRPC (gRPC Remote Procedure Calls [4]), which delivers a simple way of building connected sys-
tems. Using this technology we are able to define services and methods which can be called by the client-side remotely.

In the CoopMap platform, the back-end server is divided into three sub-components. The first component is responsible for receiving requests coming from users and for building and sending the respective responses. The second component is responsible for processing the requests. This processing consists of requests being decomposed in a group of calculations (which are performed in this component) and in a group of database statements, which are taken care of in the third component. This last component is responsible for the interaction with the Database.

3.2. Database
This platform uses two different databases, which are located near the back-end server. The main one, also called the CoopMap database, is responsible for saving information to be fetched by the application. The second database serves as a support for the main one. It keeps raw OpenStreetMap data, which can be extracted to update the streets from the main database.

The database management system chosen for this project is a relational DBMS called PostgreSQL. Besides PostgreSQL and its extension, we also required a tool called Osm2pgsql, which allows the conversion of OpenStreetMap data into our PostGIS-enabled PostgreSQL database.

3.2.1 OSM Copy Database
The OSM Copy database maintains a copy of the data from OpenStreetMap. This data allows the extraction of streets metadata, which is used in the CoopMap database. However, it holds information about every kind of object present in OSM in a very detailed way and requires filtering. This results in a significant amount of unused information considering this project’s goal. Our interest in this database is to extract the streets and their list of coordinates and export them to the CoopMap database.

It is impossible to maintain an always up-to-date copy of the OSM’s database because it would require frequent requests to their servers. Since OSM is a non-profitable platform, it cannot maintain high capability servers to answer all these requests. Our solution consists of downloading manually an OSM file (Portugal.osm.pbf) from a website called geofabrik [3] containing the most recent map version of Portugal.

After this step, osm2pgsql is used to import this file into the CoopMap database. Every object in the OSM platform has tags associated, which contain a low quantity of information. By defining a default_style file, we are able to import the tags in which we have interest and discard the rest. The tag (or key) highway=* is responsible for identifying any kind of road or path. To achieve our goal of keeping data about streets, we decided to include all the lines with a non-empty highway tag. Since streets can contain multiple roads or paths they will contain multiple lines as well. We denominate the lines present in a street as street lines.

3.2.2 CoopMap Database
The CoopMap database saves all the information required by the application and the back-end server.

The most critical information to be fetched in runtime is the streets metadata. Every time a user enters a new area on a map, the mobile application needs to request that area’s streets from the server. We decided to divide the streets in multiple segments. This way, the user can select one or more segments of a street before submitting a contribution. This allows a higher level of precision since the user can select small parts of a street. Regarding each segment, the respective street is stored as well as the coordinates of the segment and values of the contributions.

Using a script (bat file), we are able to generate a text file containing data from all the streets of Lisbon. This data results from the query to the previous database. The extracted text format for each street line is the following:

\[\text{osm\_id;street\_name;coordinates}\]

To achieve the desired format, the way element requires a double transformation, which is represented below:

\[\text{st\_asText(st\_transform(way, 4326))}\]

This statement is responsible for transforming the coordinates into the World Geodetic System 1984 (WGS84) and subsequently in a recogniz-
able text format. This transformation allows us to work with the traditional coordinate system, which is required when drawing the lines on the device screen. With the data in this format, we are able to populate (or update) the new database.

We have a table that stores information about all the street segments of the city. Each entry of this table consists of one street segment and keeps a street_id and a street_name, which come directly from the street lines text file. It also includes a segment_id generated during the execution of the script and it is attributed to each segment to differentiate it from others within the same street. The way keeps information about the pair of coordinates from the start and the end of the segment. Each entry also contains three arrays:

- The attributes array contains all existent attributes ids in the platform;
- The reviews array contains the average values for each attribute that originated from multiple contributions coming from users.
- The reviews_count array contains the number of contributions made regarding a certain attribute. This array is used to calculate the overall score of the street segment whenever a new contribution arrives.

3.3. Mobile Application

On the client-side, we decided to create a mobile application for the Android operating system targeting devices with Android 4.1 or higher. Developing in this OS facilitates access to a variety of open-source libraries like mapsforge [5], which facilitates with some map-related concepts like map rendering and layers management. Both Java and Gradle have plenty of support in the Android development.

The client-side application consists of a mobile approach where the user, using his smartphone, can interact with his communities. It is composed of three sub-components (Figure 2):

The Location Manager Module has the purpose of keeping track of the coordinates of the user's position. This allows some important features like the automatic positioning inside a map's instance. The Communication Module is responsible for establishing and keeping a communication channel with the server-side. The Map Manager Module is responsible for saving and updating a map's state during user utilization.

Inside a map, the user has two ways of interacting with it. Either he is visualizing the results of the community contributions or he is submitting contributions. The mobile application provides an easy way of swapping between the two modes.

In the Visualization mode, the user cannot submit any kind of information to the server. He can only observe the street segments' colors. The color system is described in Figure 3. The overall score of a segment varies between one and five where one is the worst score and five is the best. Each value is associated with a color in the figure.

In Contribution Mode, the colors of the lines are used to inform the user of the current contribution status. In this mode, the user can select multiple segments, one by one, to classify. However, he can also select two segments of the same line and automatically all the intermediate segments get selected.

When the user finishes selecting segments, he can press a button to open a Dialog where he can perform his final evaluation. If the server returns a success message, the segments become closed for newer contributions coming from this user.

4. Results & discussion

In this section, we present our evaluation of the proposed solution. The goal of this evaluation is to question if the following platform functional requirements were successfully achieved:

- Any user can create a map with a certain number of attributes and submit contributions;
- Other users can join a map and submit contributions as well;
For any user, submitting a contribution is a simple and quick process.

Besides these functional requirements, we also wanted to ensure that the results of submitted contributions are accessible to every user in a relatively short time.

The process of updating the CoopMap database can be evaluated as well to detect if it can be reproduced for larger areas.

4.1. Evaluation Methodology

All the tests were performed on a Xiaomi M1 A1 smartphone (Android 9.0) with the specifications: Snapdragon 625 CPU; 4GB of RAM. The computer used to run the database update has the following specifications: i5-7200U CPU; 8GB of RAM. This system is also responsible for hosting the back-end server during the tests with the mobile device. All tests were performed with a USB cable connection between the device and the server which means network latency was not an issue. Every test was performed 5-10 times to balance the effects of external conditions across all cases.

4.2. Single-User Benchmark

During this section, we will describe our benchmark tests, which were performed in a scenario containing a single user.

We wanted to allow user contributions to be accessible by everyone in a relatively short time. To evaluate this condition, we decided to measure the time since the button to submit the contribution is pressed until the database gets updated with the value of this contribution.

Every contribution contains an array of street segments. We have submitted contributions with 100, 250, 500, 1000 and 2000 street segments (contributions with distinct dimensions).

In Figure 4 we can notice a linear growth of the time spent to perform each kind of request concerning the number of segments.

We expect the typical usage of the application to consist of making multiple small contributions (contributions with 1-100 segments). Contributions containing higher numbers of segments may lead the user to lose track of the already selected segments and consequently may lead to wrong judgments of the spatial attributes. The average time to process a request containing 100 street segments was 406 ms, which we consider as a reasonable amount of time.

To evaluate the amount of data transferred from the user’s mobile device to the server, we measured the size of each contribution in Bytes. Figure 5 presents the different data sizes for each contribution considering the number of map attributes and the number of segments selected.

We can notice a proportional relation between the number of segments and the size of the contributions. However, there are two factors which may also influence the final size of the contribution:

- For each attribute, the contribution carries a value containing the user’s subjective evaluation. In most cases, this means that contributions with a higher number of attributes will have a bigger size compared with contributions with a lower number of attributes;
- gRPC serialization is not deterministic ([4]). This means that contributions containing the same kind and number of objects may end up having slightly different data sizes when serialized.

When the user enters the map for the first time, he can visualize the overall score of all the segments loaded in the map through the color system described in the previous section. For this feature
to be allowed, the client must first fetch this information from the server. The application contains a thread that handles all the requests related to map updates. These requests contain a reference point which is going to be used by the server to query the database. The query will select all the segments within a 300 meters radius from that reference point and will return this data to the client.

<table>
<thead>
<tr>
<th>Number of Segments</th>
<th>data size (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>332</td>
<td>15.4</td>
</tr>
<tr>
<td>388</td>
<td>18.4</td>
</tr>
<tr>
<td>445</td>
<td>20.7</td>
</tr>
<tr>
<td>530</td>
<td>25.0</td>
</tr>
</tbody>
</table>

Table 1: Example of map updates responses and their respective sizes (kB)

On Table 1, it is possible to find some information about the responses returned to the client. As expected, areas with higher road density result in a higher number of street segments and consequently in bigger requests.

4.3. Multi-User Experiment
We wanted to test if multiple users using the application at the same time would affect the overall performance of the platform. In other words, we wanted to measure how the platform scales when receiving requests from multiple users at the same time.

Our back-end server contains a queue (FILO approach) which accumulates unprocessed contributions. To perform this test, we decided to simulate a map context where multiple mock users would submit contributions at the same time for the same street segments. When processing a contribution, the server performs a read and consecutively a write in the CoopMap database. We need to ensure that the data remains consistent between the two operations. The point of submitting contributions for the same segments is to check if the server is able to process these contributions correctly avoiding data conflicts.

The conditions to run these performance tests are defined below:

- All requests were made within the same map, which contained two attributes;
- The number of segments for each contribution was generated randomly between 1 and 10 segments;
- The time was counted since the moment where the first request was sent until the last request was processed by the back-end server;
- We tested four different scenarios with four different amounts of users (100,300,500,1000);
- All requests were sent from the same mobile device.

As expected, time grows linearly as the number of users increases (Figure 6). Minor variations in the results come mainly from external processes related to Android.

4.4. Usability Test
To prove that our application is easy to use and that any user can create a map and start making contributions, we decided to perform a usability test.

We defined a group of small tasks which are described below:

1. **Create a map with three attributes:** This task includes selecting a name, a description and three attributes (each one containing a name and a description);
2. **Select 1-10 segments and submit a contribution:** This task implies changing the map mode to be able to select the segments;
3. **Visualize the results of the contribution:** This task is simple and consists mainly of returning to the Visualization mode of the map and assimilate the results of the user contributions;

These tasks represent the core of the application and are meant to be performed with a low number of mistakes.

To perform this test, we picked a group of five users with ages between twenty-two to fifty-nine years old. They also had different levels of experience using mobile applications. They were informed about the goal of the application and the detailed tasks. Because there is currently no tutorial, we also revealed the existence of the two map modes.

<table>
<thead>
<tr>
<th>Task</th>
<th>Avg Time (s)</th>
<th>Avg Number of Clicks</th>
<th>Avg Miss Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74.3</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>34.3</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4.9</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Results of the usability test
The average results are represented on Table 2. Considering the task 1, we did not count the number of clicks while the user was writing (map description, map name, attribute names and attribute descriptions). The timer started to count when the user entered the home screen. This means that he had to navigate through our menu and reach the “MyMaps” tab to create a new map. During this task, most of the time was spent filling the TextBox’s as expected. There was no noticeable difficulty found by the users.

During task 2 we noticed an issue. Some streets contain a significant amount of lines. This situation can difficult the process of choosing the desired segments.

Task 3 was, as expected, easy to perform by our users. They all observed the results of their contributions without any difficulty.

From these results, we can confirm that any user can create a new map instance and submit one or more contributions within a few seconds.

4.5. Database Update
To update the database, a .pfb file is necessary. This file contains a version of the OSM for a determined area. The process of updating the database follows two major phases:

1. The first phase consists of the extraction of the OSM data to our OSM copy database using osm2pgsql;
2. In the second one, the data from our secondary database is transformed and is used to update our current version of the CoopMap database.

Evaluating the first phase is not of our interest because we would be testing mainly the external tool osm2pgsql. It also depends highly on the file we are using to update our copy database (to perform this evaluation we used a file containing 186MB). Due to these two reasons, we are only evaluating the updating process between the OSM copy database and the CoopMap database.

To perform this test, we selected five radius resulting in different numbers of street lines (Table 3). These radius will determine the area from which we will extract our segments.

<table>
<thead>
<tr>
<th>Radius (km)</th>
<th>number of street lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1017</td>
</tr>
<tr>
<td>3</td>
<td>8202</td>
</tr>
<tr>
<td>5</td>
<td>18672</td>
</tr>
<tr>
<td>10</td>
<td>34271</td>
</tr>
<tr>
<td>20</td>
<td>84832</td>
</tr>
</tbody>
</table>

Table 3: Radius values and their respective number of street lines

From Figure 7 we can notice an exponential growth of the time concerning the radius values. Our Python scripts iterates over all the street lines and compares them to each other which creates a time complexity of $O(N^2)$. This fact explains this exponential growth when the number of segments increases. These comparisons are necessary to find lost lines and avoid repeated osm_ids.

5. Conclusions
Not every kind of spatial information is available in a map format. Some communities may find useful spatial details related to their interests. For instance, the jogging experience can be improved if the streets hold a certain number of attributes. Thus, if there was a platform where users could report their spatial judgments of the streets, other people could use it to improve their decisions.

The goal of this thesis was to provide an easy way of creating thematic maps where communities could work together to gather information not easily available. Based on the requirements defined, we developed a platform where anyone could create or join a map and classify streets according to attributes defined at map creation.

We started by processing the data coming from the OpenStreetMap platform. As the original data was in a raw format, it was not easily readable by the rest of the platform. Therefore, we decided to create a separate database (CoopMap database) containing the data in an easy-to-read format so that our platform did not spend significant time dealing with the formatting process. Consequently, we were able to develop a way of updating our database without losing contributions made for unchanged street segments.

With the help of gRPC, we were able to implement a server able to process multiple contributions efficiently, allowing these contributions to be observed at run-time at every users’ map.

In the end, we have developed a native mobile application, which delivers an easy way of interacting with the street lines provided by the Open-
StreetMap platform. The user can visualize the map data in a very intuitive way and make contributions in a separate Android activity.

6. Future Work
Our platform is composed of three separate components: database, server and client. All these components contain certain aspects which can be improved:

- **Database**: The introduction of Osmfilter [8] could improve the time necessary to perform a database update.
- **Back-End Server**: Optimizing the queue to process multiple contributions from the same map in parallel would improve the overall performance of the platform;
- **Mobile Application**: Reducing the number of segments would optimize the process of selecting the desired segments in the device.

This thesis was developed considering the context of a city. We would be able to scale this platform to larger areas by improving the database update script, which would use tiles instead of updating the whole city at once. It would be interesting to apply the concepts developed for this platform to a whole country.

References