Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Outdoor Sports Route Generation

Bachelor Thesis

Sven Dammann
sdammann@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

July 18, 2017

Abstract

In this thesis we enhance an existing algorithm for automatic route generation for different outdoor activities like running, cycling, biking, skating and hiking. Based on this algorithm, we built an Android app, which allows configuration of many settings and bigger route length by the user. We adjust the existing algorithm to allow the user to set individual starting and end points for the route. Further, we implement the route generating into an existing navigation app to provide the user with turn-by-turn navigation.

Contents

Chapter 1

Introduction

Outdoor sports activities have become more and more popular. Typical examples are running, cycling, biking, skating or hiking. While one can experience the nature by not being restricted to a route, the overall experience of an exercise can be improved by selecting a nice route. To get a good route, the performer of the mentioned activities has many different options. Firstly, one can use a nice route that one already knows. But with every iteration of that route, it can become more and more boring. Alternatively, one can also explore existing online tools and apps, which allow creating routes with maps or get routes shared by other users. This will result in good routes as these routes are tested and maybe even rated by other users. The disadvantage though is the fact, that those routes most likely will not start at the location of the performer and that they have predefined length which may not apply to the user's preference.

Besides following a predefined route, the user can also just decide during the exercise, which turn to take at the next crossing. This approach however is not guaranteed to result in the best possible route and experience for the exerciser, as this requires him to focus heavily on navigation and orientation during the activity. Also, if the user does not know the area well enough, there is a chance of getting lost.

Our app solves these problems, as it allows setting arbitrary starting and ending points, choosing a length and different parameters for the type of activity, the elevation and the view.

1.1 Related Work

This work is based on the Master Thesis *Smart Running Route Generation* by Jan Schulze [1]. In that thesis, an app is presented, which generates routes for outdoor running, where the shape of the routes are approximated with a triangle. The algorithm uses a route attractiveness model to find the optimal route. Further, a server is introduced, which runs the described algorithm.

1.2 Goals

There already exists an application with the goal to generate running routes (see Section 1.1). This thesis focuses on improving and enhancing that app with the following features.

Overhaul the user interface, include turn-by-turn navigation over headphones

The app should have turn-by-turn navigation over headphones as this is beneficial for the user by not having to look down at his phone at every crossing. That way focusing on the exercise rather than on navigation improves the user experience.

Improve algorithms to be able to generate longer routes efficiently

Until now the length of the routes to generate were limited to 12 km. The app should be able to generated routes with a length up to 100 km. This is possible due to parallelizing of the route calculation and optimized data management on the server side.

Add more user customizable parameters for the route generation

The user should be able to set:

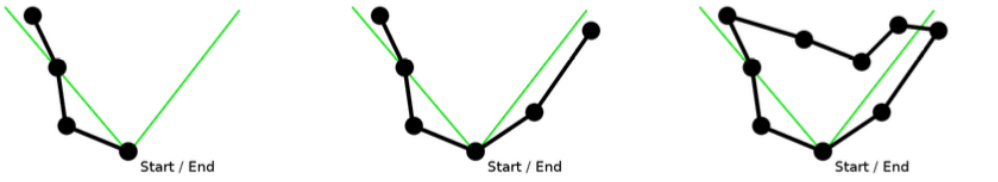
- Length: How long the generated route should be
- Type of activity: For which type of activity the user wants to generate a route
- Environment: The importance of the environment for the route
- Elevation: If the route should be steep or flat
- View: The importance of a good view
- Starting Point: Where the user wants to start his route
- Ending Point: Where the route should end

Chapter 2

Methods

2.1 Triangle Algorithm

The triangle algorithm was introduced in the past thesis by Jan Schulze [1] and is illustrated in Figure 2.1. It takes a single point as *Start* and *End* point, creates two branches in different directions and connects the two ends of those branches. This results in a triangular shaped route. Several new optimizations and enhancements to the Triangular Algorithm are discussed in this chapter.



Step 1: Create First Branch | Step 2: Create Second Branch | Step 3: Connect Branches

Figure 2.1: Triangle Algorithm, taken from *Smart Running Route Generation* [1]

2.1.1 Triangle with an *End* Point

To provide the user the ability to end the route at a different point than where it starts, we modify the triangle algorithm. To keep the triangular shape of the route, we come up with the following idea, which is graphically depicted in Figure 2.2.

First, we take a as half of the route length and c as the distance between *Start* and *End* point.

$$a = \frac{Route\,lenght}{2}$$

$$c = Distance\,between\,Start\,and\,End$$

We then calculate the midpoint $\vec{O}$ between the *Start* ($\vec{S}$) and the *End* ($\vec{E}$).

$$\vec{B} = \vec{E} - \vec{S}$$
$$\vec{O} = \vec{S} + \frac{\vec{B}}{2}$$

Then we calculate the height of the triangle where $\vec{S}$ and $\vec{E}$ form the base of it. Then we get the apex $\vec{P}$ of this triangle by going in the direction of $\vec{T}$ (orthogonal to $\vec{B}$) and length of the height h starting in $\vec{O}$.

$$h = \sqrt{a^2 - \frac{1}{4}c^2}$$
$$\vec{T} \perp \vec{B}$$
$$\vec{P} = \vec{O} + \frac{\vec{T}}{\| \vec{T} \|}h$$

Lastly, the route is calculated by generating a branch from the *Start* $\vec{S}$ to the apex $\vec{P}$ and from the *End* $\vec{E}$ to $\vec{P}$.

2.1.2 New Route Length Limit

We increase the route length limit from 12 km to 100 km, as a cyclist or a biker may not be satisfied with a route that is only 12 km long. This change alone would result in much bigger calculation times for a route being longer than 12 km. However, optimizations on the server side prevent such increase in computation time. See Chapter 4 and Section 5.2 for more details.

2.1.3 Randomness in every Route

For generating diverse - and thus interesting - routes, the route calculation is randomized. To "guarantee" a sufficiently good route, the implementation generates multiple routes and selects the best one of them.

Random Starting Direction and Angle

If *Start* and *End* are the same point, we choose for every route a random direction Φ between 0°and 360°and an angle φ between 30°and 120°. The first branch is created in the direction Φ, the second branch in the direction $\Phi + \varphi$.

Figure 2.2: Triangle Algorithm with *Start* and *End* point

Random Height of Triangle

If *Start* and *End* are not the same point, we cannot vary the direction or the angle, as the direction is defined by the *Start* and *End* points. As we want a isosceles triangle for our route, we also cannot vary the angle of the height of the triangle. We came up with the idea to vary the height, which will vary the angle of the two branches and create different routes. The height is multiplied by a randomly chosen number between -1.2 and -0.8 or between 0.8 and 1.2. By choosing negative numbers between -1.2 and -0.8 we have the ability to reflect the triangle at its base and get even more different routes.

2.2 Route Attractiveness Model

The route attractiveness model was also introduced in the past thesis [1]. It assigns every edge in the street network graph a weight which corresponds to the attractiveness of that particular edge. The attractiveness of an edge consists of its activity weight, the elevation weight and the view weight.

2.2.1 Activity Weights

The activity weight is defined for example by the type of the street (for example highway, living street, forest path, ...) and other properties (for example ground surface, motor vehicles allowed, ...) and models the attractiveness of a path for each activity. A cycler for example prefers roads and dedicated cycleways whereas a biker enjoys to ride on paths which are in the nature more than streets in the city. In the data from OpenStreetMap[1] every way has some tags containing information about the type of street or path and other data related to the path. For every activity we assign a different weight to every relevant tag, see Figure 2.3 for some examples of tags and which weights we assign to them. Then, we calculate the activity weight of a way by multiplying the weights of all tags of that way.

$$w_{\text{activity}} = \prod_{t \in Tags} t$$

[1] openstreetmap.org: OpenStreetMap [2]

Tags	Biking	Running	Cycling	Hiking	Skating
bicycle:dedicated	5.0	0.0	20.0	0.0	5.0
highway:track	20.0	20.0	1.0	20.0	1.0
tracktype:grade4	3.0	1.0	-1.0	5.0	-1.0
natural:forest	10.0	5.0	0.0	5.0	0.0

Figure 2.3: Example for some tags and their weight

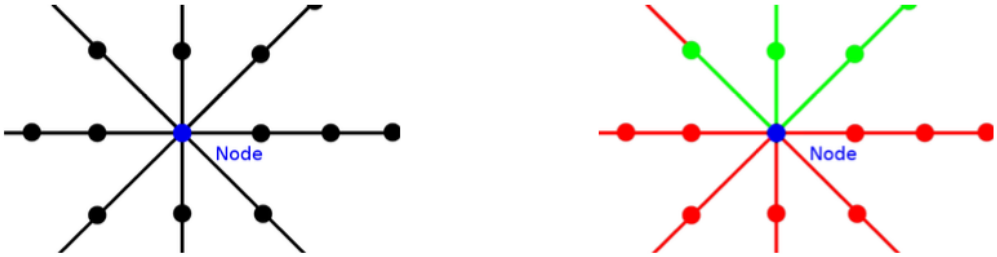
2.2.2 Elevation Map

For the elevation weights we cannot take data from OpenStreetMap[1], as they only provide hardly any topographic data. Instead, we take the data from the Shuttle Radar Topography Mission [3]. We take the absolute elevation between two points as the elevation weight for the way which connects these two points.

2.2.3 View Weights

The better the view of a route is, the more attractive is this route. A hiker for example wants to enjoy the view on his trip rather than just see a few meters ahead. To model a good view, we took a similar approach like in the past thesis [1]. We look again at the topographic data and calculate the view weight by "looking" in eight different directions. We iterate over the nodes in each direction and count the number of iterations n_d until a node has a bigger height than the node in the previous iteration. The better the view, the bigger is n_d and thus the bigger the view weight w_{view}. If there is no view, then the view weight $w_{view} = 1$.

$$w_{view} = 1 + \sum_{d \in Directions} (1 - \frac{1}{1 + n_d}) \tag{2.1}$$



We check how far the view from a certain node is in eight directions.

In this example we have a view in three directions represented by the green edges.

Figure 2.4: Example of the view calculation, taken from *Smart Running Route Generation* [1]

2.2.4 Path Attractiveness Formula

To sum up all these three weights and consider the user's preference for each of them, we come up with the following formula for the attractiveness of a way.

$$A = p_{\text{environment}} * w_{\text{activity}} + p_{\text{elevation}} * w_{\text{elevation}} + p_{\text{view}} * w_{\text{view}} \tag{2.2}$$

Where $p_{weight\ type}$ is the users preference for that weight and $w_{weight\ type}$ is the calculated weight for that way.

Chapter 3

Android App

We rework the app *Smart Route* from the past thesis [1] and perform several improvements to the user experience. A new setup page for the route parameters, a new map interface and turn-by-turn navigation are added to the application. In order to get a route, the user runs through the setup page, then sets the *Start* and *End* point of his choice and finally triggers a route generation on the server with a press of a button. Only few seconds later the route is displayed on the map and the exercise is ready to begin.

This improved version of the app is published on the Swiss Google Play Store[1]. Currently, all the route generation is performed on the dedicated server. Running the triangle algorithm on the phone is currently not implemented. However, this is theoretically possible, see Section 6.1 for more details.

3.1 Route Parameters Setup

A new setup page, depicted in Figure 3.1, allows setting all the route parameters, except the *Start* and *End* point, in one place. The user has the following options to customize the parameters.

- Route Length: To set the desired length of the route, the user can set the distance with a seek-bar at the top or tap on the distance below the bar to manually enter a number.
- Type of Activity: Depending on the activity, the route attractiveness computation uses different edge weights, as explained in Section 2.2. The following activities are supported:
  - Biking
  - Running
  - Cycling

[1] http://play.google.com/store/apps/details?id=sd.smartroute: Smart Route

  - Hiking
  - Skating

- Environment: Sets the environment weight preference $p_{environment}$
  - Insignificant: $p_{environment} = 0$
  - Slightly important: $p_{environment} = 1$
  - Fairly important: $p_{environment} = 2$
- Elevation: Sets the elevation weight preference $p_{elevation}$
  - Flat: $p_{elevation} = -1$
  - Insignificant: $p_{elevation} = 0$
  - Steep: $p_{elevation} = 1$
- View: Sets the view weight preference p_{view}
  - Insignificant: $p_{view} = 0$
  - Slightly important: $p_{view} = 1$
  - Fairly important: $p_{view} = 2$

3.2 Map Interface

The map interface shows the OsmAnd map, which is based on OpenStreetMap data (see Figure 3.2). In order to get the OsmAnd map, we implement the setup page as well as the route generation functionality of the previous app [1] into the open-source maps and navigation app *Maps & GPS Navigation - OsmAnd*[2].

The gear-wheel button in the lower left corner opens the setup page. In the lower right corner, there are two flag buttons which allow setting the *Start* and *End* point.

3.2.1 Setting *Start* and *End* Point

When the setup is completed, one can set the *Start* point by tapping on the green flag icon in the lower right corner. By doing so, the user is presented with 3 options:

- Choose Start on Map: By selecting this, the user can tap anywhere on the map to set the *Start* point.

[2] https://play.google.com/store/apps/details?id=net.osmand

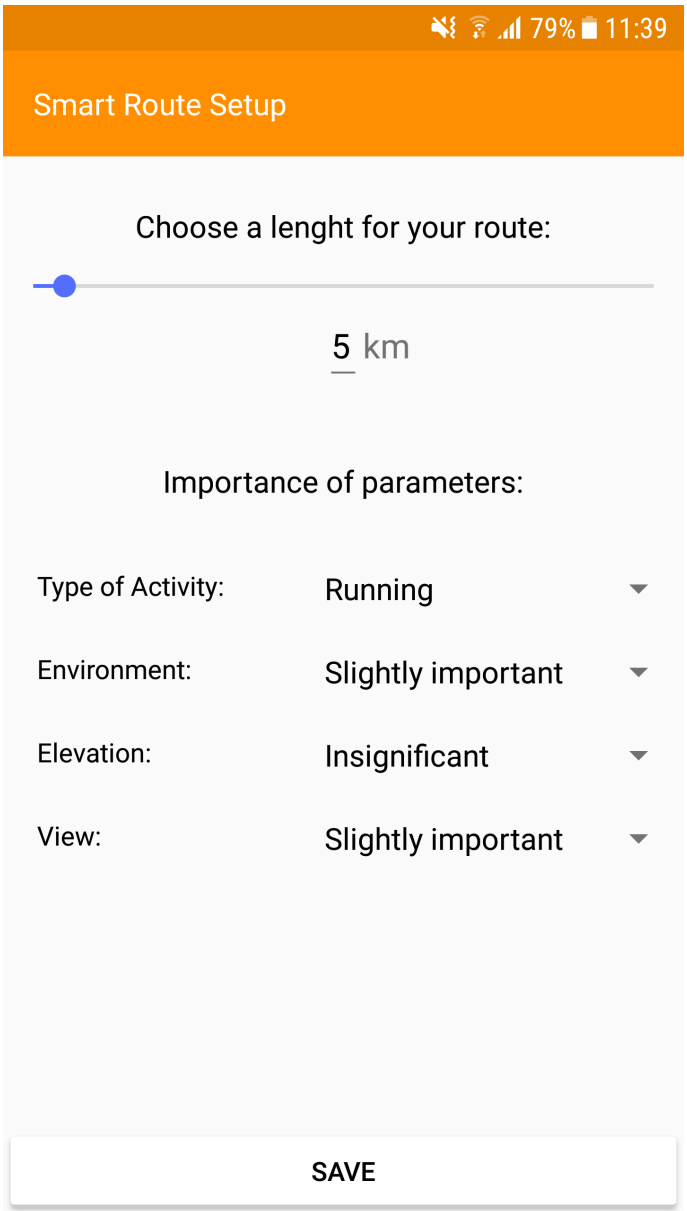


Figure 3.1: Setup Page Overview

Figure 3.2: Map Interface. Buttons (left to right): Setup page button, Start turn-by-turn navigation, Focus current location, Set *Start* and *End* point

- Use current location: This option waits for a GPS signal and then sets the start point to the current location of the user.
- Same Start as End: If the *End* point is already set, this sets the *Start* point at the same location as the *End* point.

To set the *End* point, the process is analogous to setting the *Start* point after tapping the checkered flag button in the lower right corner.

3.2.2 Turn-By-Turn Navigation

For implementing turn-by-turn navigation, we tried at first to implement the route generation into a Google Maps application. This did not work out as Google does not provide the necessary APIs to developers. In a second attempt we choose to use the open-source maps and navigation app *OsmAnd*, which has the necessary APIs. *OsmAnd* provides on-screen turn-by-turn navigation instructions (see Figure 3.3) as well as spoken audio guidance over the speaker of the phone or attached headphones.

Figure 3.3: Turn-By-Turn Navigation

Chapter 4

Server Implementation

At the moment the route generation algorithm runs on a server. We use the server that was introduced in the past thesis [1]. A limitation of that server from the last project was that only routes up to 12 km could be generated. Several changes to our server are performed in order to generate longer routes more efficiently.

In a first step, the data crawler generates the route attractiveness data for a defined region, in our example for Switzerland. This data is stored in files on disk. When a route of length L is requested via the servlet, the route generator loads all the data which is in the radius $\frac{L}{2}$ around the *Start* point. Then the input parameters are used to generate routes out of which finally the best one is returned to the servlet and from there to the application on the user's phone. This work flow is depicted in Figure 4.1.

4.1 Parallel Computing

The triangle algorithm creates multiple routes and then selects the best one of them. As the previous server only had one single core, all work was performed sequentially. With the new implementation, parallelization is used to speed up the route computation and thus allow longer routes to be generated. We define the computation of a single route as a task, which can run in a thread pool. When a route is requested by the user, a fixed number N of tasks is generated and run in the thread pool. N depends on the requested route length L:

- $0 \text{ km} < L < 10 \text{ km}$: $N = 400$
- $10 \text{ km} \leq L < 25 \text{ km}$: $N = 200$
- $25 \text{ km} \leq L < 50 \text{ km}$: $N = 50$
- $50 \text{ km} \leq L \leq 100 \text{ km}$: $N = 25$

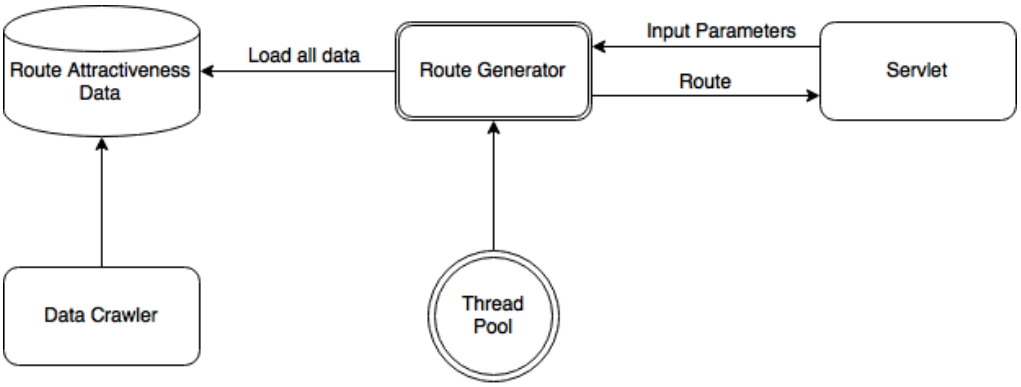


Figure 4.1: Server Diagram

After all tasks have completed, the generated routes are compared to find the best route. This is done sequentially as it would cause data races if performed in parallel. However, the comparison takes only a short time in respect to the generation of the routes.

4.2 Data Management

When a route is requested, the algorithm loads the data around the *Start* point from the disk. For routes over 10 km this takes a few seconds to complete. So with every route request, this is wasted time, especially if the route is generated at the same *Start* point over and over again. We overcome this inefficiency by loading all available data once at start-up of the server and keeping it in the RAM. Now, the algorithm has faster and lower-latency access to the needed route attractiveness data which results in faster route generation times.

Chapter 5

Evaluation

In this chapter, we analyze computation times and several example routes for each activity and the influences of different weights for various activities.

5.1 Route Length

First, we examine the accuracy regarding the route length of the algorithm. We generate five routes each for different lengths and look how close the returned route to the requested length is. We plotted the results in a scatter graph, which is depicted in Figure 5.1. As we can see in the graph, the returned route length is always, except in the case of 100 km, inside a 10 % margin of error, which we assume is acceptable.

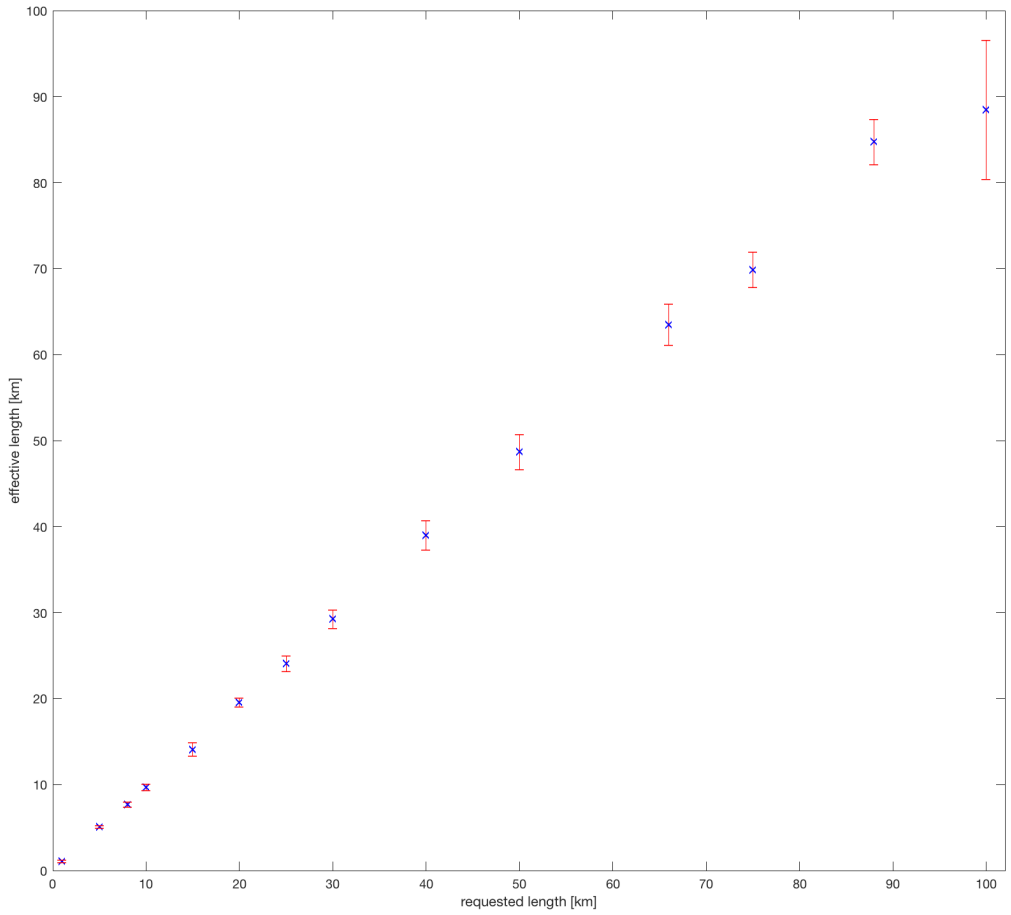


Figure 5.1: The requested length L in km on the x axis versus the effective length in km on the y axis

5.2 Calculation Time of Routes

Next, we take a look at calculation times of the triangle algorithm. For this evaluation we again generated five routes each for different lengths and plotted the results in a scatter graph. The graph is depicted in Figure 5.2 and shows that the computation times are always shorter than 5 seconds for routes under 75 km.

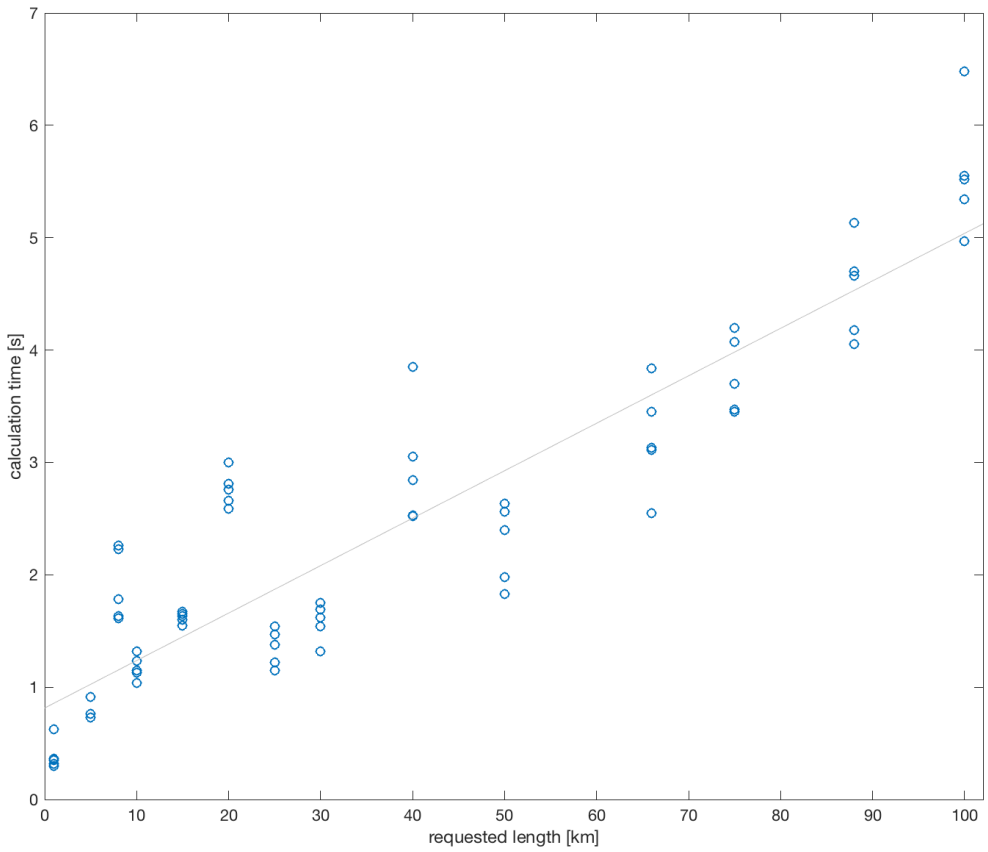


Figure 5.2: The requested length L in km on the x axis versus the computation time T in seconds on the y axis, best linear fit $y = 0.0422 * x + 0.8141$

5.3 Activity Differences

To evaluate the variation between the different activity settings, we generated five routes for each activity and analyze the proportion of forest, streets and non-streets (paths, tracks, forest tracks). The first route for each activity had its starting point near the forest at the Zürichberg, the second route started in downtown Zürich, the third one in Zürich Altstetten again close to the forest, the forth in Zürich Schwammendingen and the last one in Rümlang. We omitted the hiking sample in downtown Zürich, as we think that this is not a realistic scenario. Instead, we generated an additional route for hiking at the Uetliberg. Figure 5.3 depicts the average composition of all generated routes.

In the best case the forest and tracks proportion for Cycling and Skating would be 0. However, as we can see in the results, Cycling and Skating have very low forest and tracks proportions as in some cases the algorithm cannot

Activity	∅ Forest [%]	∅ Streets [%]	∅ Tracks [%]	Screenshot
Running	48 ± 25.8	38 ± 29.5	62 ± 29.5	Figure A.1
Biking	40 ± 33.8	49 ± 28.2	51 ± 28.2	Figure A.2
Cycling	3 ± 4	85 ± 17.6	15 ± 17.6	Figure A.3
Hiking	60 ± 30.3	26 ± 16.6	74 ± 16.6	Figure A.4
Skating	2 ± 4	92 ± 2.4	8 ± 2.4	Figure A.5

Figure 5.3: Example for some tags and their weight

find a route without going through a forest or taking a track. Yet, they have high proportions of streets, which is beneficial for these activities. Running and Biking have a good proportion of forest in their routes and take about equal proportions in streets and tracks. Hiking clearly prefers forest and tracks over streets, which makes sense.

5.4 Inaccuracy of Path Tracking

For scalability reasons, the route attractiveness data does not include every single node from the OpenStreetMap data. Nodes which only have two neighbours and thus only connect their two neighbours, are omitted. This results in path tracking inaccuracies, as depicted in Figure 5.4.

Figure 5.4: Inaccuracy of Path Tracking example. The red circles illustrate the missing nodes, and the red lines show the optimal tracking.

Chapter 6

Conclusion

In this thesis we presented several improvements to the triangle algorithm, which allow generating routes for five different outdoor activities and with different *Start* and *End* points. Further, we introduced a new path attractiveness formula with user preference factors for each weight which improved the flexibility of generated routes.

The existing Android app was reworked and enhanced with a new setup page for all the route parameters, a new map interface based on *OsmAnd*, as well as turn-by-turn navigation for the generated routes.

On the server side, we introduced parallel computing and a new data management, which allows faster route generation and more efficient calculation of routes longer than 10 km.

6.1 Future Work

Client Side Computation

If the user has no cellular data plan or wants to generate a route in an area without reception, the ability to do client side computation of routes would be beneficial. While it is theoretically possible to run the triangle algorithm on a phone itself without a huge effort in code rewriting, one would have to develop an optimized data management of the attractiveness data for Android phones. The current implementation requires around 10 GB of RAM for all the data of Switzerland, which as of today no mobile phone supports. One possible approach would be to partition the data of Switzerland into different regions, for example every canton of Switzerland as one region, and save each region in a zip file. Then, the app could download the necessary zip file and run the route generation on the phone. Nowadays, nearly every phone has at least two CPU cores, so we could even benefit from the parallel computing when doing client side route generation.

Population Density for View Weights

Currently, the view weights only take topographical data into consideration. In a city however, there could be a good view according to the weights where in reality there is not because of many houses and buildings blocking the view. The current implementation does not work as expected, as there is no noticeable difference between routes with a good view compared to routes without view. One possible solution to create view weights closer to the real world view could be to find a data source of the population density in the current region and divide the weights by this density. This would result in very low view weights inside of big cities and reasonable big view weights in free nature and thus making them more realistic.

Dynamic Route Update

One further feature which could be implemented in the future is dynamic route updating. When the user leaves the generated route in the current implementation, the app tries to navigate him back onto the computed route. Leaving the route can happen by mistake or consciously by the user. In the later case, it could be desirable to ask the user if he wants a new route with his current location as *Start* point and with the length of the remaining part of the old route.

Data Optimizations

The data as well as the data management could be optimized further. As mentioned above, all the data of Switzerland requires about 10 GB of RAM. We could not figure out why this is the case, since on disk the data takes roughly 2.5 GB. This would require memory analysis and maybe some data structure optimizations. In addition more data could be generated, for example throughout Europe. Also, one could try to fix the inaccuracy issue mentioned in Section 5.4, as this would improve the navigation experience in some cases.

Bibliography

[1] Schulze, J.: Smart running route generation. (November 2016)

[2] Haklay, M., Weber, P.: Openstreetmap: User-generated street maps. IEEE Pervasive Computing **7**(4) (2008) 12–18

[3] Rabus, B., Eineder, M., Roth, A., Bamler, R.: The shuttle radar topography mission-a new class of digital elevation models acquired by spaceborne radar. ISPRS journal of photogrammetry and remote sensing **57**(4) (2003) 241–262

Appendix A

Route Types Examples

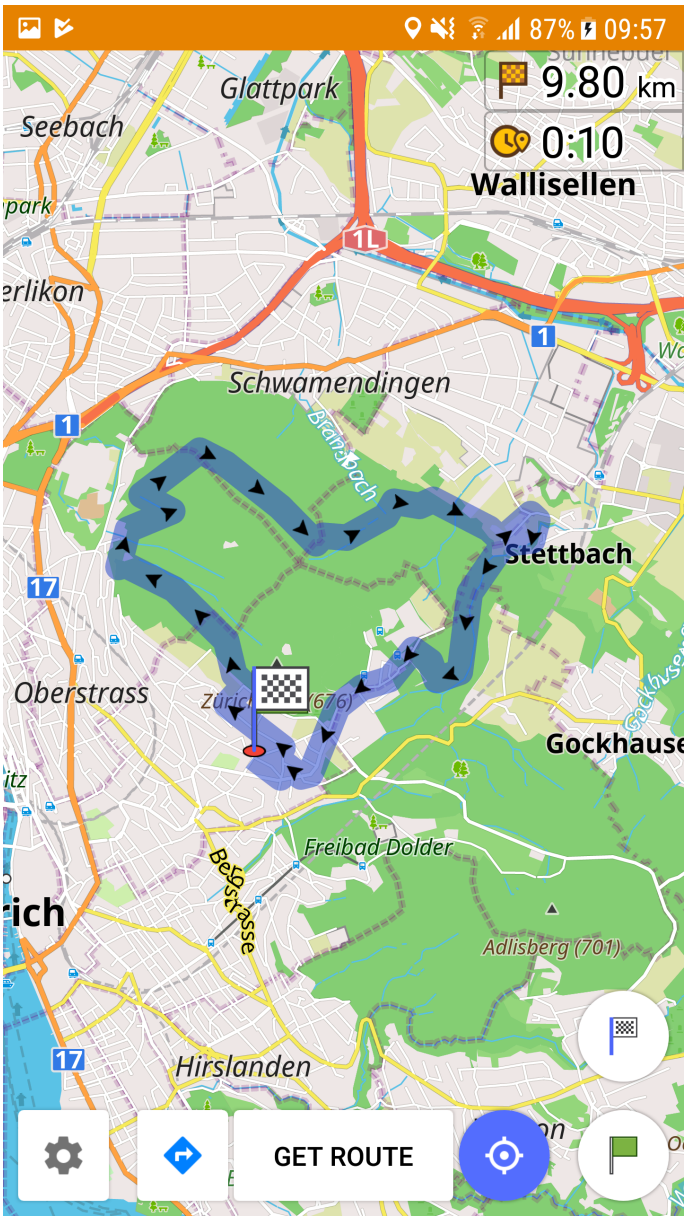


Figure A.1: Running Example Route

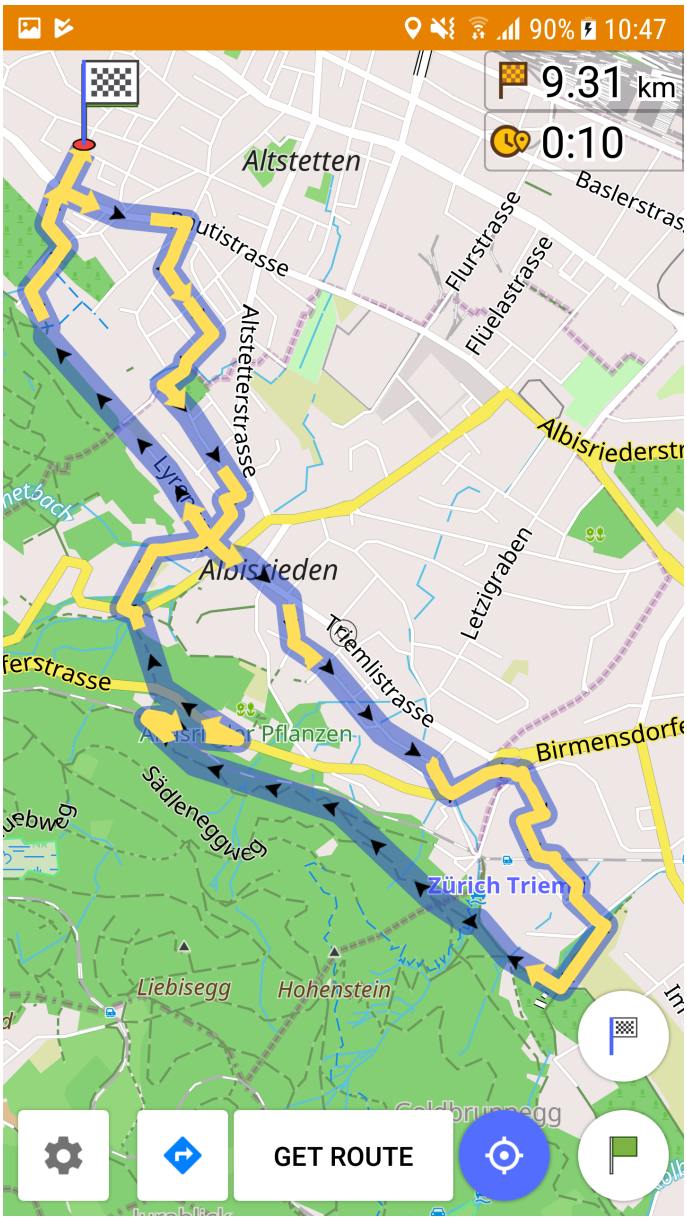


Figure A.2: Biking Example Route

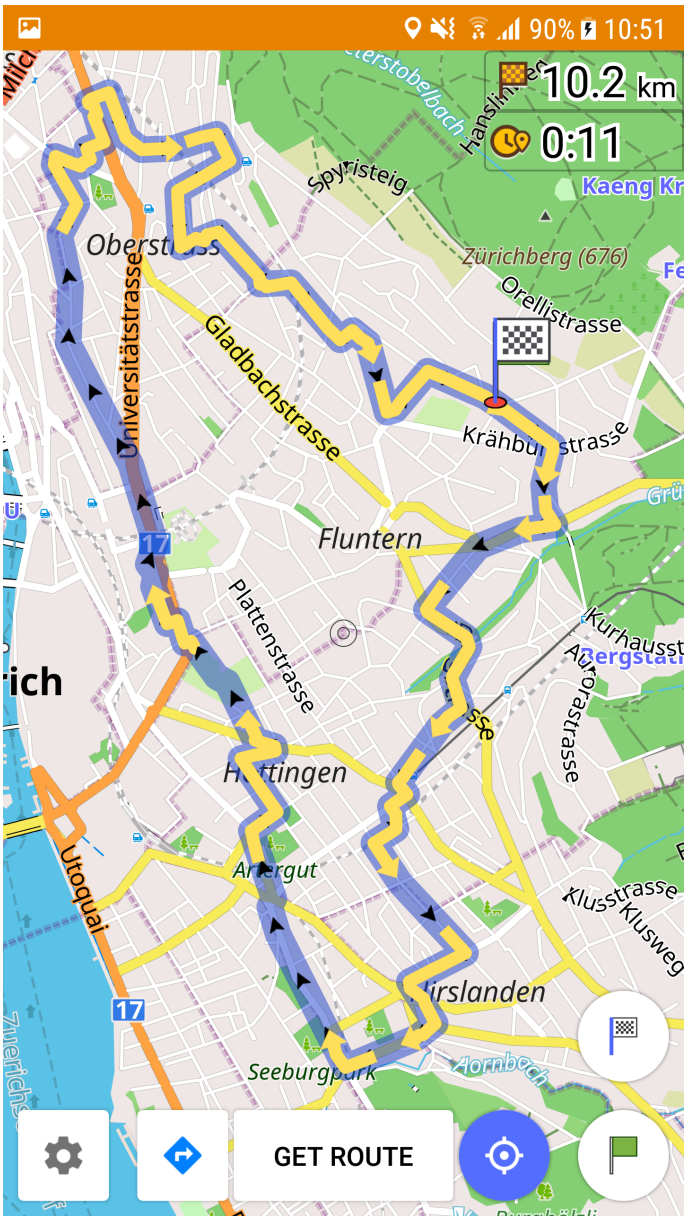


Figure A.3: Cycling Example Route

Figure A.4: Hiking Example Route

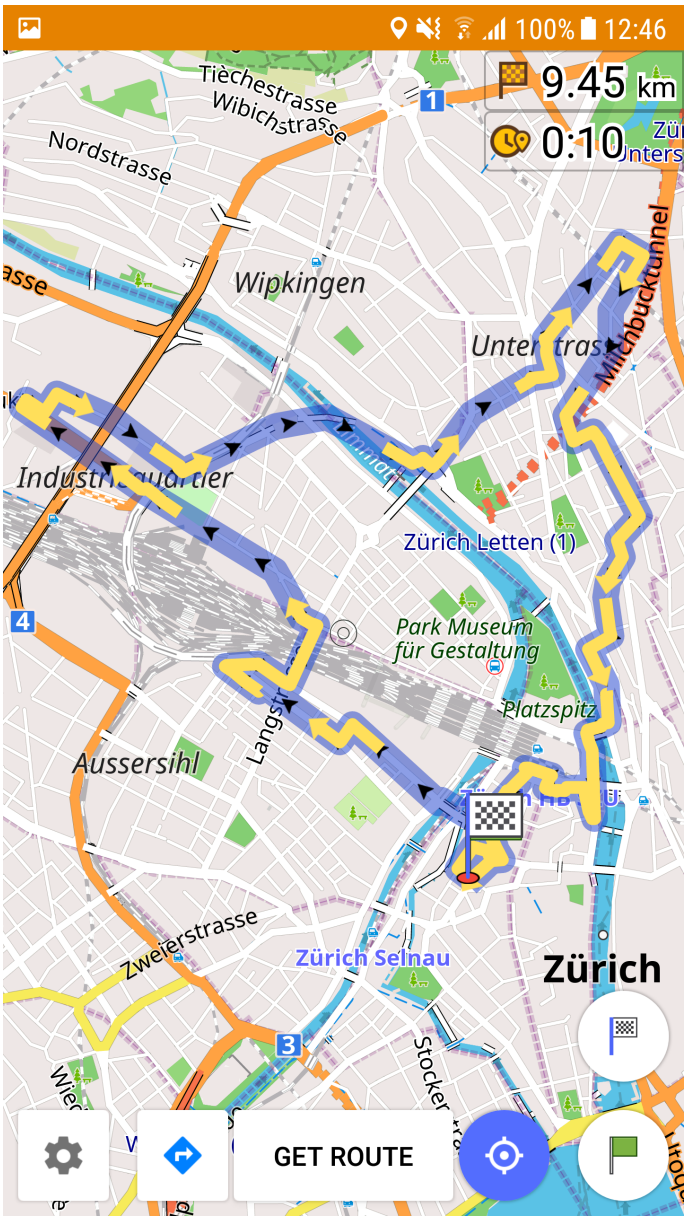


Figure A.5: Skating Example Route