



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

*Distributed
Computing*



Attractive Sports Route Generation

Semester Project

Feiyu Jia

`jiaf@student.ethz.ch`

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:

Manuel Eichelberger

Prof. Dr. Roger Wattenhofer

May 30, 2018

Acknowledgements

I would like to thank my supervisor Manuel Eichelberger for his amazing ideas and technical guidance during the period of this project. I also want to thank my family and friends for their support whenever I needed it.

Abstract

In previous theses, a website called *SmartRoute* is made to produce worldwide routes for different activities according to users' needs and preferences. In this work, improvements are made to the website to make it more user friendly and easier to use. Furthermore, more metrics which improve the route quality are included, and a mechanism for map database updating is implemented to keep the necessary data up to date.

Contents

Acknowledgements	i
Abstract	ii
1 Introduction	1
2 Frontend	2
2.1 New Functions	2
2.2 HTTPS Certificate	4
2.3 Responsive Website	4
3 Backend	8
3.1 Weather Information	8
3.1.1 Weather Gathering and Storage	8
3.1.2 Expected Effect on Route Generation	10
3.2 Route Shape	12
3.3 Data Updating	14
3.3.1 Database Writing Mechanism	14
3.3.2 Updating Schedule	14
4 Results	15
4.1 Performance Measurements	15
4.2 Quality of Routes	20
4.2.1 Weather Effects	20
4.2.2 Reshaping Algorithm Effects	23
5 Discussion and Future Work	27
Bibliography	28

CONTENTS

iv

A Declaration of originality

A-1

Introduction

When doing sports, people have their personal expectations towards routes according to activities and local topographical features. They may prefer to their daily routes in the neighbourhood, but it is possible that they get bored one day and want to try a new route which also meets all their expectations, or when they relocate to a new place it is more convenient for them to just use a recommended route instead of exploring one by themselves.

To offer a competitive solution for generating sports routes, a website named *SmartRoute* has been developed in the Distributed Computing Group, it offers users different activity routes according to their desired start/end point, length and preference towards specific metrics. Thanks to previous theses' work [1, 2, 3], *SmartRoute* could already let the user choose one activity from biking, running, cycling, hiking and skating, set the desired start/end points and length of route and indicate how much they care about the importance of environment, view and elevation. And such routes could be got in almost all over the world as shown in the last thesis [1].

In this project, the website user interface is enhanced to be more user friendly and tidy, since the previous version is just a prototype and seems a bit complicated for fresh users. Chapter 2 explains detailed changes made to user interface, newly added functions and the design of responsive website which makes *SmartRoute* more attractive.

Chapter 3 contains some changes made to the backend. First, a dynamic factor—weather information, is now taken into consideration during route generation and helps to improve the quality of the recommended route, with the use of information got from Open Weather Map [4]. Second, there is a discussion about the route's shape and how to reduce the number of zigzags in a route. Third, a mechanism is introduced, which aims at updating the map database regularly to keep the map data up to date.

Chapter 4 shows results of the speed and success rates performance of *SmartRoute*, and a quality comparison between routes generated before and after adding weather factor and route reshape respectively.

Frontend

The previous website interface is shown in Figure 2.1. It can be used in any browser as long as it supports HTML5 and JavaScript [1]. However, this user interface still has some problems: the Open Street Map may look unfamiliar to users, the selection of start and end locations may confuse users since these are not shown on the map, and the display can not adapt according to the browser's size. To make the website interface more user-friendly and attractive, plenty of improvements have been introduced.

2.1 New Functions


First of all, the Open Street Map is covered by a Google Map's tile [5] in the *SmartRoute* website, it makes the website looked more familiar to users, since Google Map is more popular than Open Street Map.

Then, some control functions [6] have been added into the map, including: fullscreen, mouse position, scale line, zoom and zoom slider. With the use of these, users can choose to see a full screen map, know the accurate coordinates of the location when they move the mouse over the map, have an intuition about the real distance between the aimed start and end point, and see map details or a thumbnail.

It is also helpful to know a user's current position since it helps propose nearby routes. In order to do that, the geolocation function [7] is used to find the user's location. Then the map centre is adapted to that point. However, most browsers forbid the obtainment of user's location using HTML5 geolocation API from pages delivered by non-secure connection, like Hyper Text Transfer Protocol (HTTP). To use geolocation in the *SmartRoute* website and increase the *SmartRoute* website security, the web server is upgraded to support Hyper Text Transfer Protocol Secure (HTTPS). More details are discussed in Section 2.2.

Some symbols are added to mark the position of start/end point in order to help users have a direct feeling about the chosen locations. A red \triangle is used to

SmartRoute Proof of concept



The screenshot displays a map of Central Europe, including parts of France, Germany, and Switzerland. Major cities like Mulhouse, Freiburg, Basel, and Zurich are labeled. The map shows a network of roads and geographical features like lakes and rivers.

Below the map, there are several interactive elements for route configuration:

- Selecting on Map:** Radio buttons for Start and End.
- Start Latitude:** **Start Longitude:**
- End Latitude:** **End Longitude:**
- Activity:** (with a dropdown arrow)
- Desired length in km:** A slider bar with a value of
- Importance of Environment:** A slider bar with a value of approximately 10%
- Importance of Hills:** A slider bar with a value of approximately 20%
- Importance of View:** A slider bar with a value of approximately 30%
-
-

Figure 2.1: A screenshot of the previous *SmartRoute* website

mark the start point, while a red ∇ is used to mark the end point, these two triangles are merged into a red hexagon if the user chooses to make the end point coincident with the start point. Moreover, a "Clear Map" button is added to clean these symbols and the generated route when the user wants to get a new route with different parameters.

Besides, the weighting of different parameters during route generation may be confusing for fresh users, so brief explanations are shown. Users can see them when the mouse crossed the different importance in laptops' browsers, or click to see in mobile phones.

The setting for environment importance is put in the backend of *SmartRoute* with a default value to use moderate importance, since it is strange to ask users how much they want to do this activity after a specific activity is selected.

An example of the modified frontend displaying a running route with different start and end point in the vicinity of Zurich is shown in Figure 2.2. Another example displaying a cycling route with the same start and end point in the vicinity of Geneva is shown in Figure 2.3.

2.2 HTTPS Certificate

Hyper Text Transfer Protocol Secure (HTTPS) is a secure version of Hyper Text Transfer Protocol (HTTP). It encrypts communication between the user's browser and the web server. For the *SmartRoute* website, a Secure Sockets Layer (SSL) certificate which supports the use of Transport Layer Security (TLS) protocol is chosen to use.

When a HTTPS connection is required, the *SmartRoute* website initially sends its SSL certificate to user's browser, then the SSL handshake is initiated between the browser and the website. This handshake generates shared secrets to establish a unique secure connection between the browser and the website.

In order to make *SmartRoute* support the use of geolocation and ensure the security of users' private data, the web server is upgraded from HTTP protocol to HTTPS protocol, and a SSL certificate is qualified from *Let'sEncrypt* [8]. Besides, a user's permission is necessary when the geolocation is used in browsers.

2.3 Responsive Website

To make the website adaptive to different screen sizes and a variety of devices, responsive design is used for suitable display methods for different display sizes. The user interface design is done with CSS. Several display plans are defined according to the screen size. Besides this, related positions of buttons or sliders also change corresponding to this size. The website may look differently in

SmartRoute

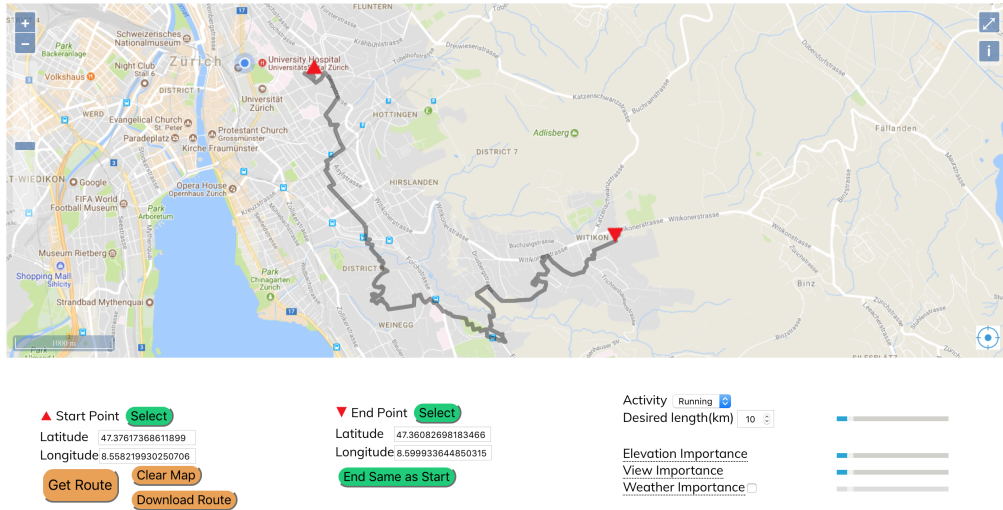


Figure 2.2: A screenshot of the frontend showing a running route in the vicinity of Zurich

SmartRoute

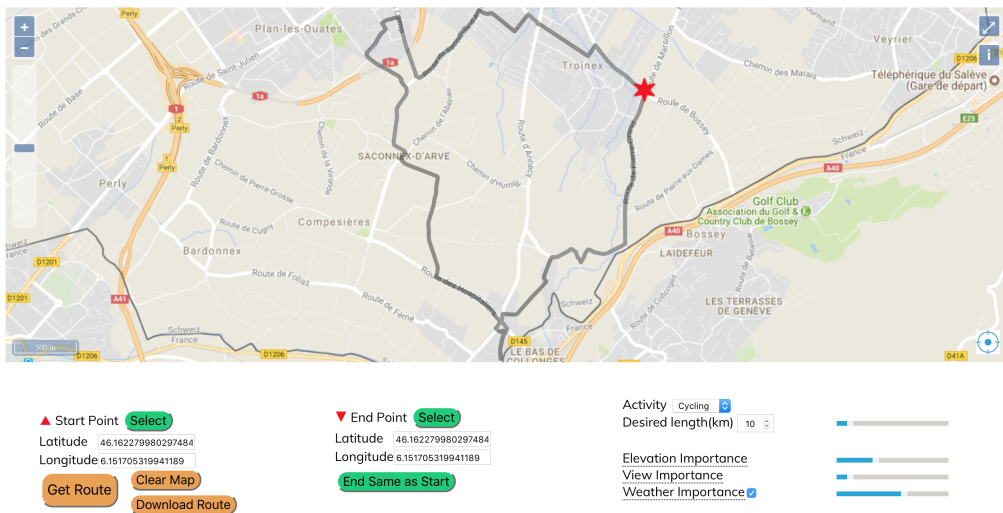


Figure 2.3: A screenshot of the frontend showing a cycling route in the vicinity of Geneva

various browsers or operating systems. However, it is verified that *SmartRoute* looks nice in several popular browsers, including Firefox, Chrome and Safari. Figure 2.4 shows how the website looks like when the size of browser becomes smaller. Figure 2.5 shows how the website looks like in mobile phones.

SmartRoute

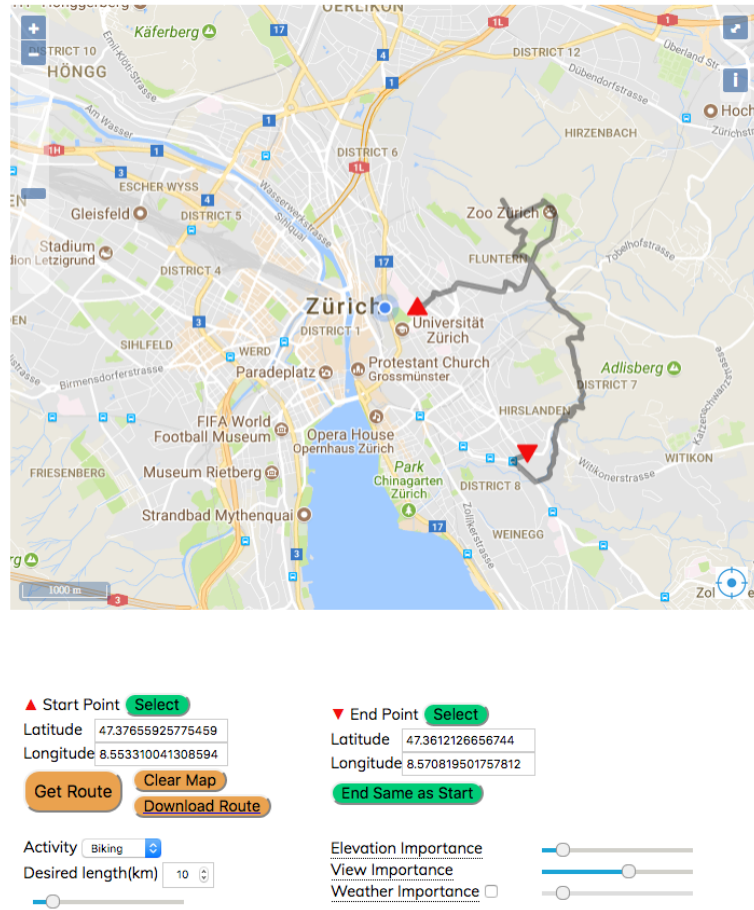


Figure 2.4: A screenshot of the *SmartRoute* website with smaller browser's size with Firefox

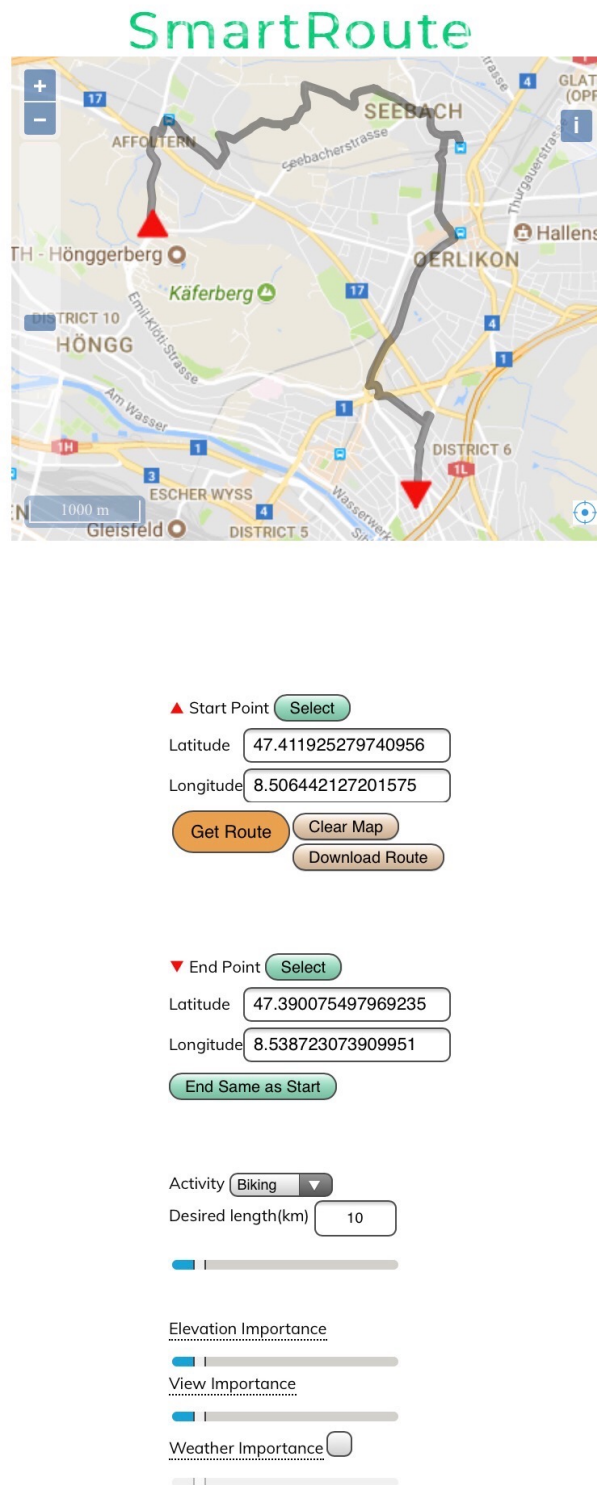


Figure 2.5: A screenshot of the *SmartRoute* website on a mobile phone with Safari

Backend

3.1 Weather Information

The previous version of *SmartRoute* does not take dynamic situations into consideration, for example, weather conditions. However, varying weather modes or temperatures may affect users' preferences to routes. Besides, some parameters may become more or less significant in specific weather modes. In this section, weather information gathering, storage, and usage are discussed.

3.1.1 Weather Gathering and Storage

The Java API of Open Weather Map [9] is used to get current weather mode and temperature information from Open Weather Map [4], which is a global geospatial platform offering worldwide current weather conditions and forecasts. Since the maximum number of obtained weather information is constrained to 60 requests per minute for each API Key, it is unrealistic to get weather information for each user request. Instead, a database storing weather information of 10761 positions is deployed on the server to gather weather information per latitude/longitude degree on the continents once per hour. When the user needs to generate a route, weather information whose gathered location is nearest to the route's start point is used.

Weather information is gathered from places where *SmartRoute* can be used and the weather is not almost the same during the whole year. In that case, the latitude's scope is between -70° to 70° , and some regions are excluded, like Mideast and some tropical islands. When routes need to be generated in these areas, no weather information is taken into consideration. In order to get the coordinates of the land instead of the ocean, the Matlab function *landmask* [10] is used to determine if places given specific coordinates are land or ocean.

To differentiate weather information's gathering time and location, each of them needs a unique ID. A 12 digits number is designed to represent it, the first six digits represent the number of hours between current time and midnight,

January 1st, 1970, the 7th and 8th digits are the absolute value of the position's latitude, the 9th to 11th digits are the absolute value of its longitude, the last digit takes integer from 0 to 3 depending on signs of latitude and longitude as shown in Table 3.1. A weather information ID's example is shown in Figure 3.1.

Value	Latitude	Longitude
0	non-negative	non-negative
1	non-negative	negative
2	negative	non-negative
3	negative	negative

Table 3.1: The rule of last digit for weather information's ID

The weather information gathering task is distributed into smaller ones to get per minute in order to not exceed the data limitation, and with the use of crontab, the weather information is set time to obtain automatically in the server. Moreover, all points get the latest weather information once per hour.

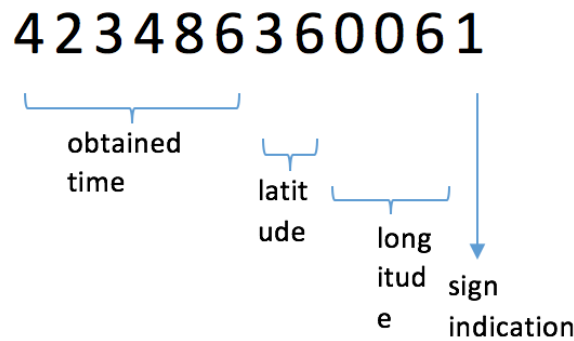


Figure 3.1: An example of a weather information's ID

After obtaining weather information, it is stored in a MongoDB database system [11] in the JavaScript Object Notation (JSON) format as shown in Listing 3.1. When the user sets parameters to generate a route in *SmartRoute*, matched weather information with closest obtained time and nearest location is retrieved from the database by the use of the rule of weather information's ID. Considering database's capacity and time effectiveness of current weather, these weather informations are stored for two days.

```
{ "_id" : NumberLong("423486360061"), "weaMode" : 721, "Temp" :
  288.08 }
{ "_id" : NumberLong("423486360011"), "weaMode" : 701, "Temp" :
  288.75 }
```

```

{ "_id" : NumberLong("423486370091"), "weaMode" : 802, "Temp" :
  288.696 }
{ "_id" : NumberLong("423486370081"), "weaMode" : 801, "Temp" :
  291.15 }
{ "_id" : NumberLong("423486370071"), "weaMode" : 800, "Temp" :
  288.546 }

```

Listing 3.1: An example of weather information encoded in a JSON document

3.1.2 Expected Effect on Route Generation

After obtainment of weather information, it is designed to optimise routes offered by *SmartRoute* in several respects.

First, two new weights – surface weight and forestry weight – are gained with the use of tags offered by Open Street Map [12] for each node [2]. Surface weight contains information about a path’s surface condition, since it is inconvenient for users if a path with muddy surface is recommended under raining or snowing conditions. Instead, the use of asphalt, paved and concrete paths is encouraged. When utilizing surface weight, historical weather modes are also taken into consideration, since the road may still be muddy because of previous day’s raining. Forestry weight contains information about whether the path is in forest or not, and its aim is to encourage users to use forestry roads when raining or extremely hot weather. Considering the joined effects of surface weight and forestry weight, routes with forestry shadow and without muddy surface are recommended if it is raining. After adding these two weights, the node information in JSON format is as shown in Listing 3.2. The usage of newly added time information is discussed in Section 3.3.1. The user can decide whether weather information needs to be counted or not, if yes, corresponding weather preference can be set, then this preference is used as preference for surface weight and forestry weight in certain weather modes.

```

{
  "_id" : NumberLong(1463979399),
  "time" : NumberLong(17638),
  "coordinates" : {
    "type" : "Point",
    "coordinates" : [
      12.3466969,
      55.6155932
    ]
  },
  "height" : 2.010454444799877,
  "neighbors" : [
    {

```

```

        "nID" : NumberLong("3389543578"),
        "bikeWeight" : 9,
        "cycleWeight" : 16,
        "elevationWeight" : 0.1376064668812846,
        "hikeWeight" : 5,
        "skateWeight" : 16,
        "viewWeight" : 1,
        "runningWeight" : 9,
        "forestryWeight" : 0,
        "surfaceWeight" : 10,
        "distance" : 31.562429428100586
    },
    {
        "nID" : NumberLong(288941213),
        "bikeWeight" : 9,
        "cycleWeight" : 16,
        "elevationWeight" : 0.11440948370598938,
        "hikeWeight" : 5,
        "skateWeight" : 16,
        "viewWeight" : 1,
        "runningWeight" : 9,
        "forestryWeight" : 0,
        "surfaceWeight" : 10,
        "distance" : 15.54236888885498
    }
]
}

```

Listing 3.2: An example of a node encoded in a JSON document

Secondly, the visibility is not so ideal in fog, raining or snowing. In this case, only a part of the view weight is used. It is set that 20% of the view weight is taken into account when it is foggy, and 50% of the view weight is used when it is raining or snowing.

Thirdly, extreme temperatures may influence a user's preference towards elevation. For example, it becomes easier to get sunstroke when the temperature is quite high. Under this circumstance, users are encouraged to use flatter paths when the temperature is above $35^{\circ}C$ to keep away from sunstroke. And it is recommended to use steeper paths when the temperature is below $0^{\circ}C$ to do more exercise to protect against the cold. Also, there will be less windy since users move slower in this case.

3.2 Route Shape

From Figure 3.2, we can see that the previous routing algorithm may produce plenty of zigzags. This is kind of annoying for users in practice, since they may have to make many sharp turns within short distances. To reduce the number of zigzags in a route, a reshaping algorithm is implemented.



Figure 3.2: An example of a route with zigzags

SmartRoute generates a route consisting of a set of nodes. Every two neighbouring nodes are joined by a line, and there is an angle between two joined lines. A large angle indicates that user needs to make a sharp turn. Those angles can be calculated by the difference of bearings, while the bearing is able to be computed given the coordinates of a line's start and end points [13].

A window with certain length l is used to constrain the observed scope to a local sub-route, then we sum up the angles within this scope. If the sum exceeds a threshold, a conclusion is made that this sub-route contains too many zigzags, and needs to be smoothed. To smooth a zigzag path, another path which can link the zigzag's start and end point more directly is searched. If such a new path exists, the old one is deleted and the route is relinked. Then, the window is shifted to next sub-route, which starts from the next node of the last sub-route, and repeated until completes the whole current route's searches. The window's length l is set to be 300 m, and the threshold of the angle's sum is 360° . Figure 3.3 illustrates the main procedures for reshaping a route.

Because the new path has a shorter distance, the final route's length may be smaller than the desired one. In this case, the difference between the real length and the desired one is checked, if it exceeds the tolerance, the path is extended. The extended place is approximately 1.5 km away from the route's end point if the route's length is greater than 1.5 km, or the end point if the length is smaller or equal to 1.5 km. In the end, it is verified that the length tolerance scope is less than 5%.

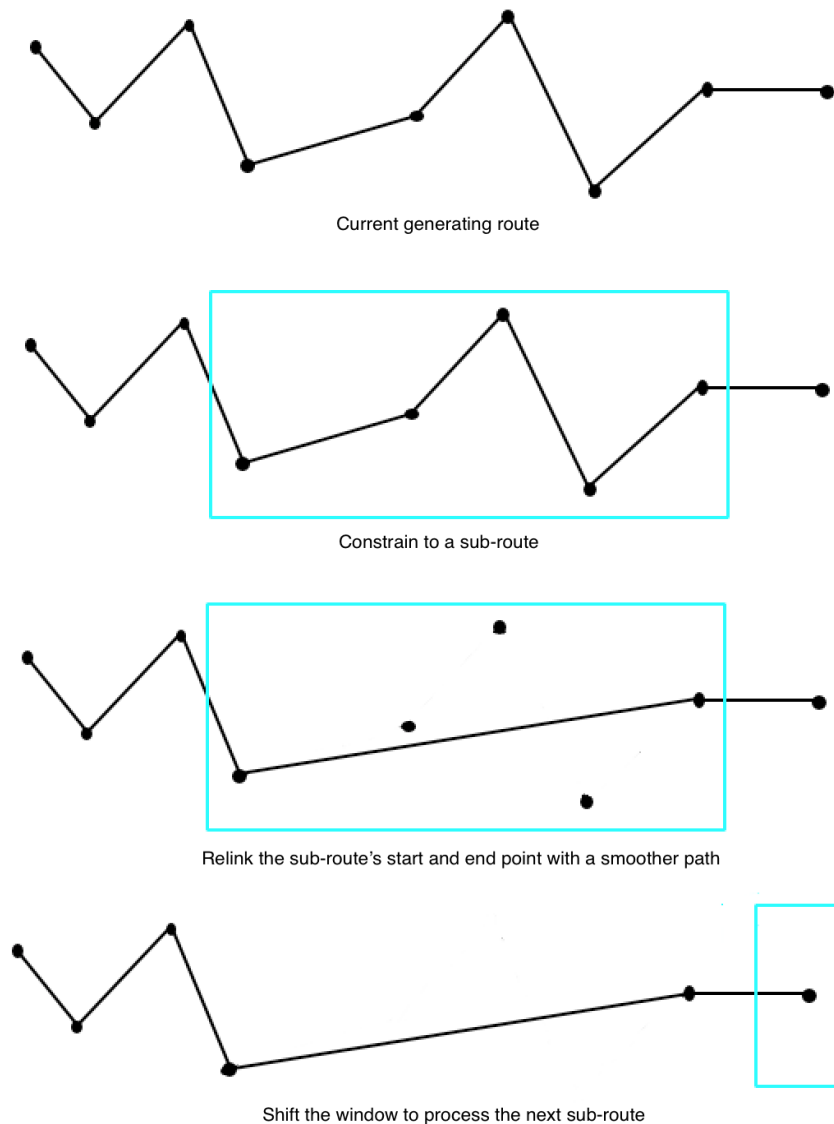


Figure 3.3: Illustration of the reshaping algorithm

3.3 Data Updating

The node information is obtained through Open Street Map [12], and Open Street Map data is updated every day. Specifically, some new nodes are added while nodes which are out of date are deleted. In order to use the up-to-date nodes' information, we update the database regularly in a back process without huge inference with database's usage.

3.3.1 Database Writing Mechanism

As introduced in the previous thesis [1], a MongoDB database system [11] is used to store all nodes' information. When a node needs to be stored, MongoDB checks whether there already exists a node with the same ID. If such a node already exists, then the node's information is updated to the current one while the old one is deleted. If not, a new node is inserted into MongoDB. With the use of this writing mechanism, we can update or add nodes' information, but if some nodes are deleted from Open Street Map, we need to find another way to remove the corresponding nodes from the database.

For this, a timestamp is added to each node, which indicates the number of days between current time and January 1st, 1970. When updating the map data, the node's timestamp is also updated to the current one if this node is still used in the latest map. If a node's timestamp is not updated after one iteration of the map data's updating, we delete it from the database.

3.3.2 Updating Schedule

Because of the database updating's huge needs for memory resources, the speed of route generation slows down if the database's updating works at the same time with route generating. In order to make sure the database updating does not interfere with *SmartRoute* route generating tasks, the updating can proceed during the night. Also, crawler tasks for each continent are separated into smaller ones to complete, each of them contains crawler tasks for some countries. It is verified that every task can be finished during the night or only uses a small amount of memory resources during the day. The tasks of downloading the latest map data are also made into subparts, because the data source – *Geofabrik* [14] – set a data download's limitation to allow downloading approximate 50 files at one time.

With the use of crontab, the update runs automatically in the background. One iteration of updating is done every 3 months. All areas are updated at the same frequency, which is identical to Open Street Map [12].

4.1 Performance Measurements

In order to measure the performance of *SmartRoute*, the same randomized benchmark from the previous thesis [1] is used. To avoid fine tuned test datasets, random points are chosen with the help of a random coordinate generator – *GeoMidpoint* [15]. An example of chosen points is shown in Figure 4.1, it can be seen that some points are in the ocean. The aim of including these points is to test the speed of returning an empty route when no route exists.



Figure 4.1: An example of 150 random points selected within 1000 km of Zurich

SmartRoute recommends the best route within k candidate routes. The time of generating a route linearly increases with the value k . In order to get a route with high quality as soon as possible, the effect of k is tested. 30 random points

within 100 km of Zurich are chosen as the start/end point of the route. Tests are divided into four groups according to the desired length, that is, smaller than 10 km, between 10 km and 25 km, between 25 km and 50 km and larger than 50km. In each group, several desired lengths are selected, and several values of k are tested for each length. The upper bound of k is manually set to avoid spending long time to return a route. All five activities have been tested for each k and each desired length. The quality of routes is quantified by the average route’s weight per meter. Because of that, only successful responses are counted. Results are shown in Figure 4.2.

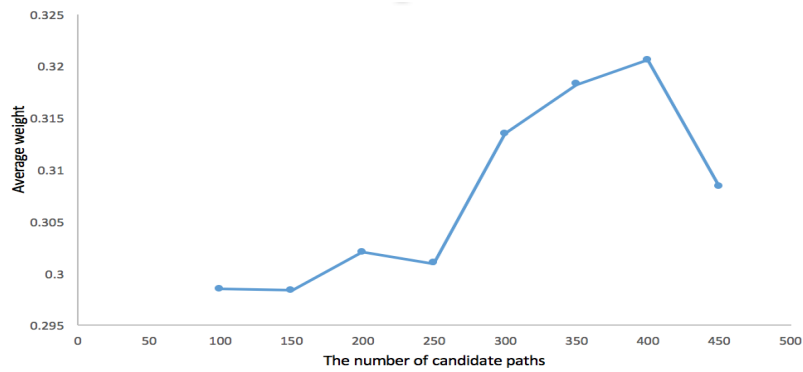
After analysing the result of Figure 4.2, the value of k is chosen to be the inflection point or the point at which the average weight growing speed slows down. That is 350 for a route whose length is less 10 km, 200 when length is from 10 km to 25 km, 30 for a route with desired length between 25 km and 50 km and 15 if the route’s length is larger than 50 km. It can be seen in Figure 4.2 that the average weight does not always increase with the growing of candidate paths’ number, the reason is that *SmartRoute* contains some random factors, for example, one part of routing algorithm is *RandomPoints* [1]. Although each datapoint in Figure 4.2 is the mean of hundreds of tests, it still cannot avoid the bias caused by random factors.

After setting the value of k , the success rates for different activities are tested. 30 random points are generated within 100 km of Zurich, each of them is set to be start and end point for a 15 km route. Table 4.1 shows the number of requests which can return a recommended route for each activity. It can be seen that hiking and running have the highest success rates while skating only returns 7 non-empty routes. This is because footpaths suitable for hiking and running are able to be found almost everywhere, but it is difficult to find usable routes for skating which have both high quality surface condition and light traffic.

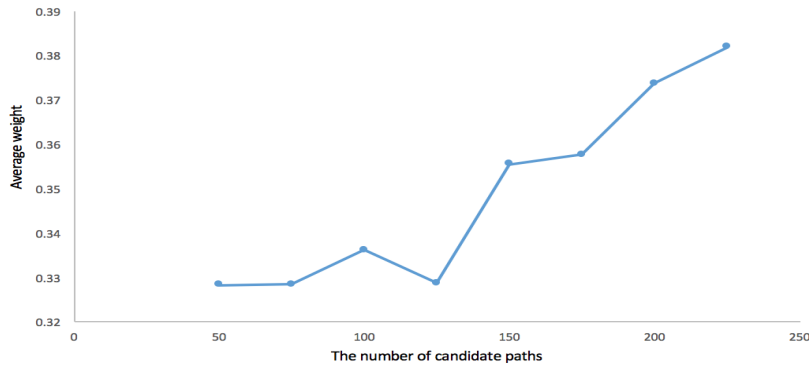
Activity	Requests	Successful Requests	Success Rates
Biking	29	16	55.17%
Running	29	18	62.07%
Cycling	29	15	51.72%
Hiking	29	21	72.41%
Skating	29	7	24.14%
Total	145	77	53.10%

Table 4.1: Summary of successful requests for different activity

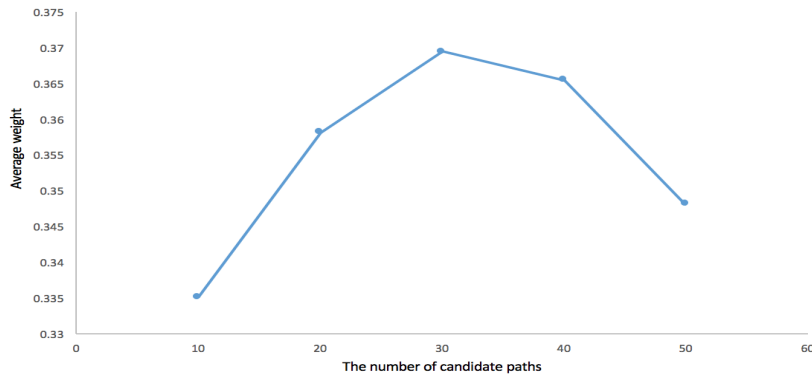
In general, there are several possible reasons that *SmartRoute* returns an empty route or cannot response successfully. First, the start or end point is not on the land, for example, the start or end point is in the ocean or lake. Second, the start or end point is selected in an area where has no path nearby. Third, there is no suitable path for a specific activity, for instance, skating needs paths



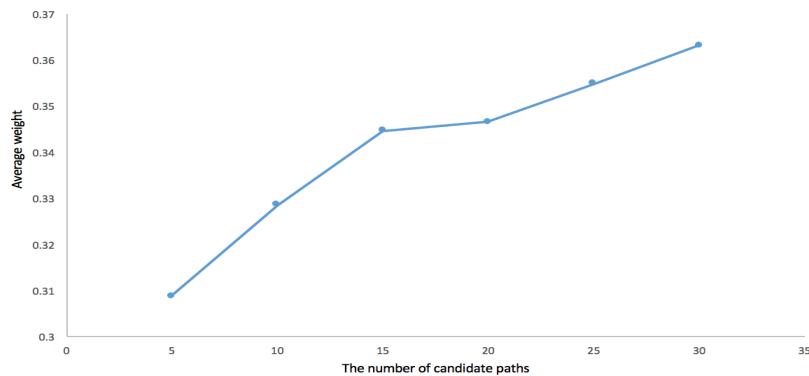
(a) A line chart for desired length less than 10 km



(b) A line chart for desired length between 10 km and 25 km



(c) A line chart for desired length between 25 km and 50 km



(d) A line chart for desired length larger than 50 km

Figure 4.2: The recommended route's average weight with different numbers of generated candidate paths

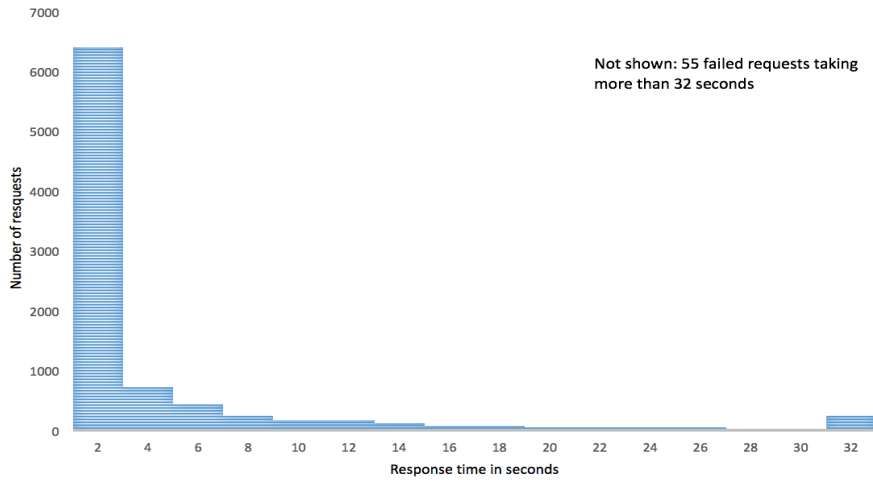
with high quality surface condition, and sometimes it may be difficult to find one. Fourth, *SmartRoute* sets penalty for particular routes, like *SmartRoute* never recommends to use private routes. Fifth, a long route tends to use loops, however, since *SmartRoute* forbids using repeated paths in local area for better user experience, it is not easy to find one which meets almost all requirements in long distance and does not contain too many loops.

Next, tests are run in different geographic regions. We select 150 random points within 1000 km of Zurich, 200 random points within 2500 km of Denver and 100 random points within 1000 km of Brasilia respectively. Random points are used as start and end points of the route, the route’s length is set to be 5 km, 15 km, 30 km and 50 km, and the activity is chosen from biking, running, cycling, hiking and skating. A histogram of the response time is shown in Figure 4.3. There is a peak around 30 s in Figure 4.3(a) and 4.3(c), the reason is that the timeout in this test is set to be 30s, then most failed requests returns empty route messages. Table 4.2 shows a comparison of success rates and average time between different geographic areas. Among the 8940 requests, 55 failed requests take more than 32 seconds, 10 of them happen within 2500 km of Denver, while the rest of them are within 1000 km of Zurich. The possible reason for *SmartRoute* spending more time to generate routes in Europe is that the map data is denser in Europe than in other regions.

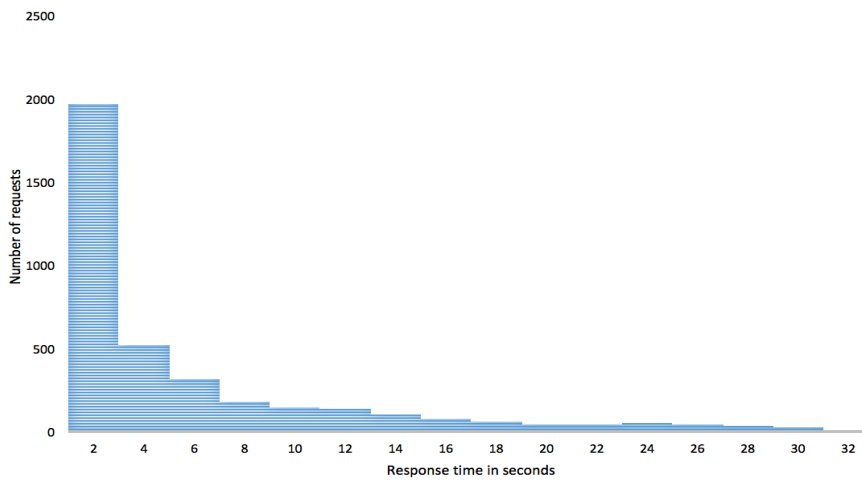
Location	Requests	Successful Requests	Success Rates	Average Time[s]
Zurich	2980	1386	46.51%	7.10
Denver	3980	1403	35.25%	2.15
Brasilia	1980	934	47.17%	1.17
Total	8940	3723	41.64%	3.58

Table 4.2: Comparison of success rates and average request time in different geographic regions

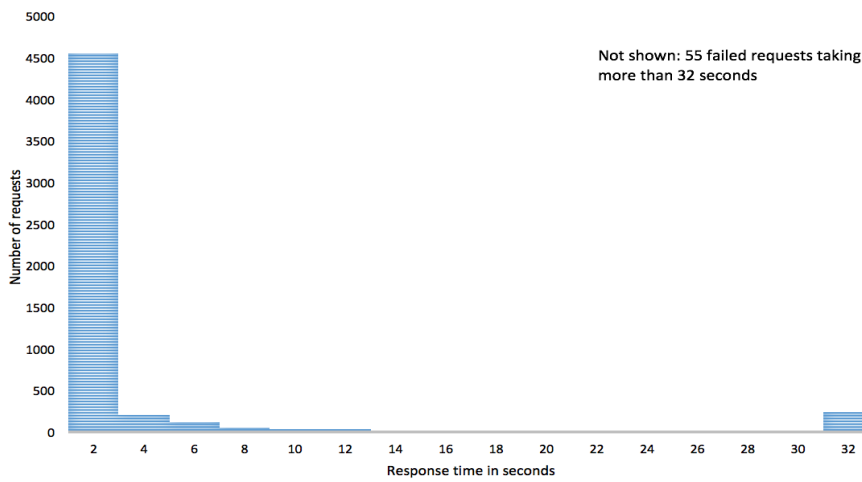
Then, *SmartRoute* is made to generate routes with long distances. The start and end points of routes are 150 random points selected within 1000 km of Zurich. The desired lengths are 80 km and 100 km respectively, and the activities are chosen from biking, running, cycling, hiking and skating. In order to give *SmartRoute* enough time to generate a long route, the allowed timeout is changed from 30 s to 60 s in this test. In practice, the timeout adapts to the desired length, in order to offer *SmartRoute* more time to generate long routes. Table 4.3 shows the corresponding success rates and average time. It can be seen from Table 4.3 that the success rates are low, this means that to really get a route this long, one may have to wait quite long time in expectation.



(a) A histogram for all requests' time



(b) A histogram for successful requests' time



(c) A histogram for unsuccessful requests' time

Figure 4.3: Response time for routes with length of 5 km, 15 km, 30 km and 50 km, activity of biking, running, cycling, hiking and skating acquired in 450 random points

Length	Requests	Successful Requests	Success Rates	Average Time[s]
80 km	745	213	28.59%	31.36
100 km	745	140	18.79%	36.92

Table 4.3: Success rates and average request time when generating long routes

4.2 Quality of Routes

The quality of routes is hard to be quantified, since a good route should consider all parameters which are set by users and have a nice shape, and there is no specific criteria for them and can only be observed manually or measured from single respect. In this section, some routes are generated in selected areas to compare the effects of weather information and the reshaping algorithm.

4.2.1 Weather Effects

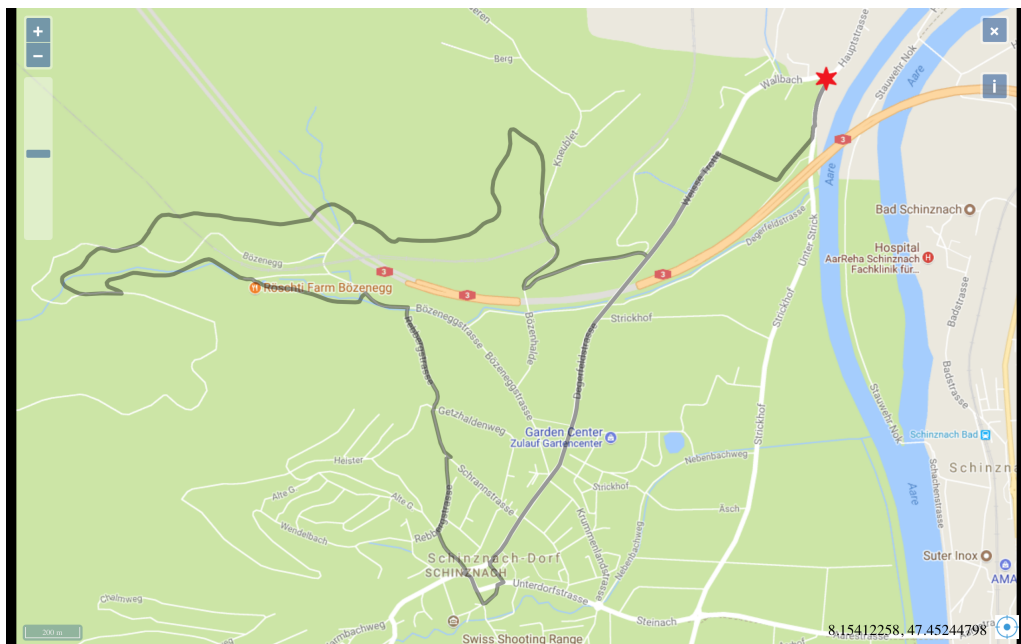
To test whether weather information can affect route generation in *SmartRoute* as expected, several tests are run under manually set weather modes or temperatures. In these cases, all parameters except the compared factor are the same within each group.

When the weather condition is raining or extremely hot, forestry paths should be encouraged to use as discussed in Section 3.1.2. There is an example under extremely hot condition as shown in Figure 4.4. Figure 4.4(a) shows the route when the weather importance is 0, in other words, no weather information is taken into consideration during route generation, and forestry paths are not encouraged to use specifically. Figure 4.4(b) shows the one when the user indicates a great preference towards weather condition, forestry paths are highly encouraged to be used when weather mode is extremely hot. It can be clearly seen that forestry weight helps the user to choose routes with shadow under particular weather conditions.

If the weather condition is raining or snowing, *SmartRoute* encourages the user to use paths without muddy surface. Figure 4.5 shows two running routes generating with or without weather preference under snowing condition respectively. Figure 4.5(a) shows when the weather is not a considered factor during route generation, no preference towards paths' surface is shown. In this case, the route's average surface weight is 0.137. Figure 4.5(b) shows when the weather is a significant factor during route generating, *SmartRoute* helps the user to choose paths with good surface conditions. The average surface weight of the recommended route is 0.189, it means this route contains more paths with good surface conditions, like asphalt, paved and concrete, the result can reach the intended objective discussed in Chapter 3.1.2.

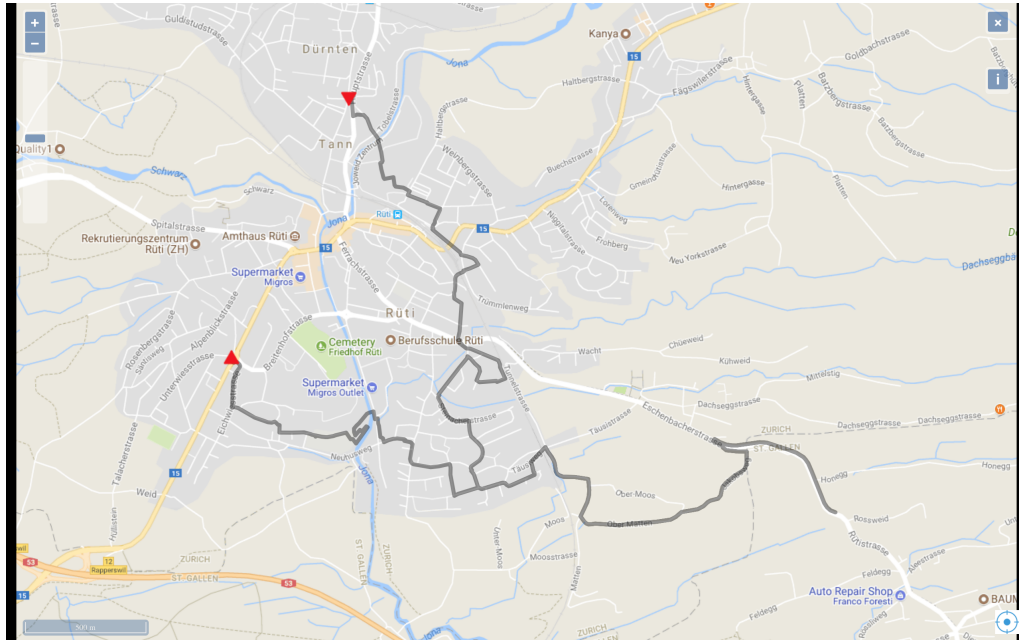


(a) A running route generating without weather preference under extremely hot condition



(b) A running route generating with high weather preference under extremely hot condition

Figure 4.4: An example of forestry weight's effect



(a) A running route generating without weather preference under snowing condition



(b) A running route generating with high weather preference under snowing condition

Figure 4.5: An example of surface weight's effect

In specific weather condition, weather mode makes difference to the view weight as discussed in section 3.1.2. Figure 4.6 shows an example that different percentages of view weight are taken into consideration, and Table 4.4 lists the average view weight per meter for each route in different weather mode. Figure 4.6(a) shows the route generating under fog, while only 20% of view weight is included in this circumstance. Figure 4.6(b) shows a route when there is snowing, at this time, half of original view weight is counted. Then, if the weather has no negative effect on field of vision, view weight is 100% counted, a result is shown as Figure 4.6(c). It can be seen that specific weather modes can weaken the effect of view weight as desired.

Weather Mode	Foggy	Snowing	Clear
Average View Weight	2.326	2.634	2.791

Table 4.4: Route’s average view weight in different weather mode

Temperature also makes influence to route’s preference as listed in Section 3.1.2. Figure 4.7 shows three routes generating under three different temperatures. If the temperature is lower than $0^{\circ}C$, *SmartRoute* encourages to use steeper routes to keep warm as shown in Figure 4.7(a). In contrast, if the local temperature is too high, flatter paths are more frequently used during route generation, as shown in Figure 4.7(c). The default case is when the temperature is between $0^{\circ}C$ and $35^{\circ}C$, no influence caused by temperature works on the elevation importance, Figure 4.7(b) illustrates that case. To offer an intuitive comparison, Table 4.5 lists the three routes’ average elevation weight per meter respectively. From previous thesis [2], it can be known that the elevation weight is proportional to absolute elevation between two points, in other word, higher elevation weight indicates steeper paths. As shown in Table 4.5, *SmartRoute* encourages using steeper paths at low temperature, and discourages the use of them at high temperature as desired.

Temperature	$-10^{\circ}C$	$20^{\circ}C$	$40^{\circ}C$
Average Elevation Weight	0.675	0.595	0.529

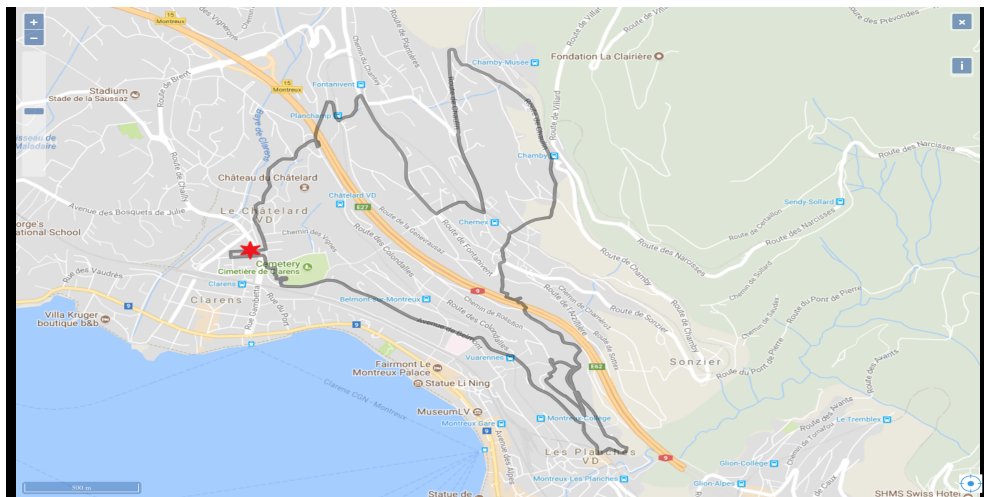
Table 4.5: Route’s average elevation weight at different temperature

4.2.2 Reshaping Algorithm Effects

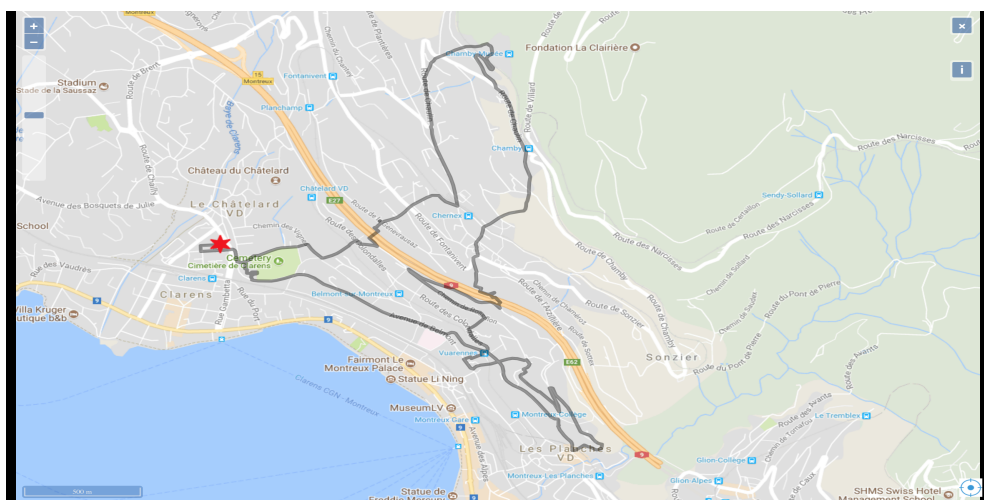
As illustrate in Section 3.2, a reshaping algorithm is implemented to reduce the number of zigzags in recommended routes. Figure 4.8 shows two routes generated with or without use of the reshaping algorithm. All other parameters are the same in this example. It can be seen that some zigzags are deleted and nodes are linked by smoother paths.



(a) A running route generating under foggy condition

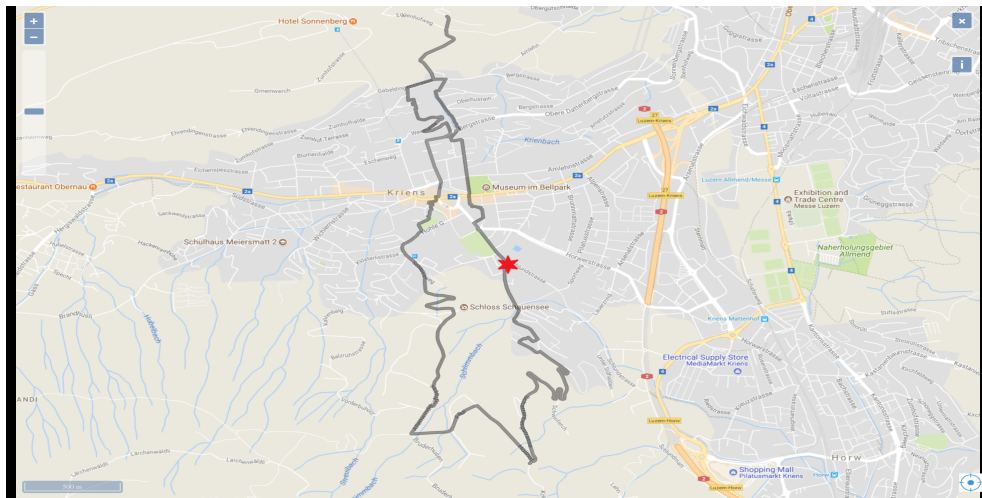


(b) A running route generating under snowing condition



(c) A running route generating under clear condition

Figure 4.6: An example of weather mode's effect towards view weight



(a) A running route generating when temperature is low

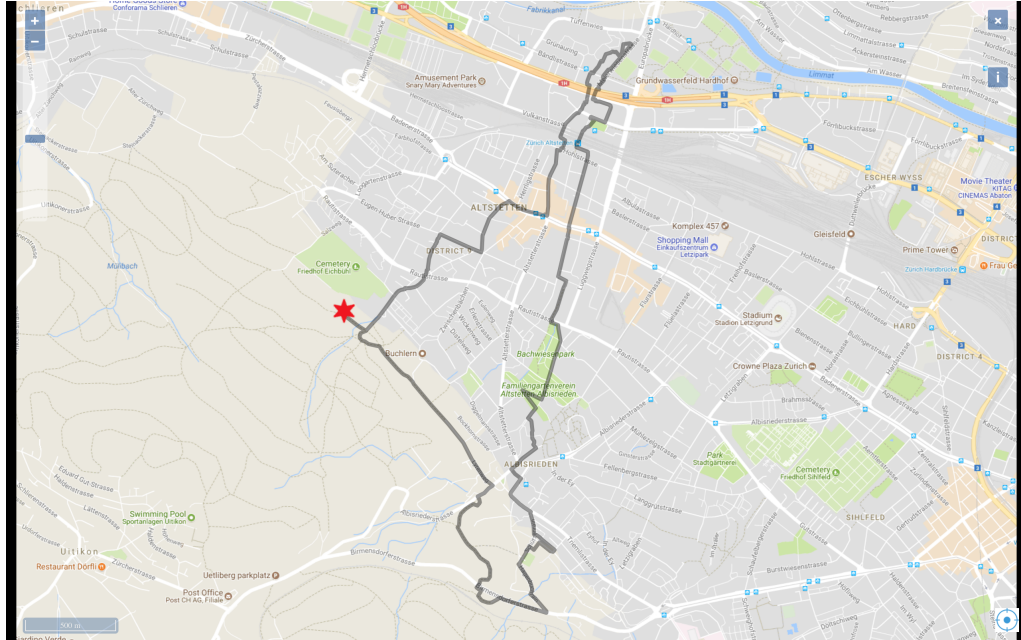


(b) A running route generating when temperature is moderate

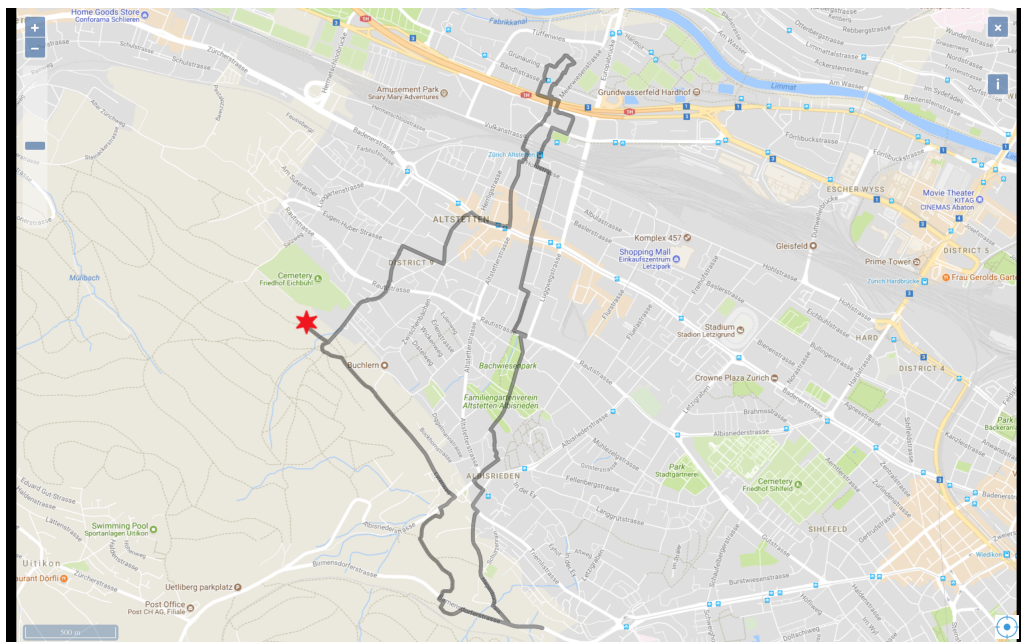


(c) A running route generating when temperature is high

Figure 4.7: An example of temperature's effect towards elevation weight



(a) A running route generating without reshaping algorithm



(b) A running route generating with reshaping algorithm

Figure 4.8: An example of reshaping algorithm's effect

Discussion and Future Work

Due to the limitation of the web server's memory resources, the weather's effect towards view weights is just simply represented by different percentages of view weight. Although more rigorous way is to calculate several view weights with different visibility and store in the database, then decide to use which one according to current weather mode. With more potential memory in the server, this way can be used to replace the current one.

About the shape of route, a reshaping algorithm is implemented to reduce the number of zigzags. Currently the window length is set to a fixed number. However, an idea is to make the length adapt to the average distance between two neighbouring nodes within one area, since the density of nodes is varying in different places. Although the window can help constrain the scope to a local area to optimise the shape, the global shape should also be considered, since sometimes there may be a case that sub-route within a small area is nice for users, but the conjunction between two adjacent paths forces the user to take a sharp turn.

And there are still plenty of improvements which could be done about a route's shape. *SmartRoute* generates routes by connecting several sub-routes. This opens the possibility that a node may be used several times in one route if it appears in several sub-routes, although repeated nodes have been forbidden to use when generating sub-routes. New methods could be found to reduce the number of repeated nodes in one route.

In the current version of *SmartRoute*, user feedback has not been taken into consideration yet, but it could be useful to get user's feedbacks and recommend routes according to their historical routes and interests. Or another metric like path's attractiveness or popularity could be added as an attractiveness weight to each node in the database.

Moreover, there is the case that a user deviates from the purposed route in practice. A dynamic method may be used to adapt routes according to a user's current location.

Bibliography

- [1] Weinbuch, J.: Worldwide Sports Route Generation. Semester Thesis, ETH (2017)
- [2] Dammann, S.: Outdoor sports route generation. Bachelor Thesis, ETH (2017)
- [3] Schulze, J.: Smart running route generation. Master Thesis, ETH (2016)
- [4] Open Weather Map Contributors: Current Weather Data. <https://openweathermap.org/current/> Accessed 2018-05-30.
- [5] Google Map Contributors: Google Map. <https://www.google.com/maps/> Accessed 2018-05-30.
- [6] Open Layers Contributors: OpenLayers Control API. <http://openlayers.org/en/latest/apidoc/ol.control.Control.html> Accessed 2018-05-30.
- [7] Open Layers Contributors: OpenLayers Geolocation API. <http://openlayers.org/en/latest/apidoc/ol.Geolocation.html> Accessed 2018-05-30.
- [8] Let's Encrypt Contributors: Let's Encrypt. <https://letsencrypt.org> Accessed 2018-05-30.
- [9] Singh, A.K.: Owm Japis. <https://bitbucket.org/aksinghnet/owm-japis/> Accessed 2018-05-30.
- [10] Greene, C.: Matlab Function Landmask. <https://ch.mathworks.com/matlabcentral/fileexchange/48661-landmask/> Accessed 2018-05-30.
- [11] MongoDB Contributors: MongoDB. <https://www.mongodb.com/> Accessed 2018-05-30.
- [12] Open Street Map Contributors: Open Street Map. <https://www.openstreetmap.org/> Accessed 2018-05-30.
- [13] John M.: Calculating a bearing between points in location-aware apps. <https://software.intel.com/en-us/blogs/2012/11/30/calculating-a-bearing-between-points-in-location-aware-apps> Accessed 2018-05-30.

- [14] Geofabrik Contributors: Geofabrik. <http://www.geofabrik.de> Accessed 2018-05-30.
- [15] GeoMidpoint Contributors: GeoMidpoint. <http://www.geomidpoint.com/random/> Accessed 2018-05-30.