Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

GEOFABRIK
neogeografie // software // beratung

Accessibility analysis for emergency service vehicles

Supervisors:
Prof. Freckmann and Prof. Günther-Diringer

Benedikt Futterer
matriculation number 54856

08. September 2017

I Abstract

In this work a website should be developed to perform accessibility analyses for non-police emergency services using only open source software and open data, because they can be adapted to the personal needs and are free to use.

For this purpose we evaluate the suitability of OpenStreetMap as basis for the computation, as well as different open source routing algorithms. These evaluations yield that OpenStreetMap data is suitable to perform accessibility analyses, however it is still incomplete in some regions. Valhalla is identified as the most suitable routing engine.
Important terms and algorithms are explained to support the comprehension. Additionally the state of the art methods of accessibility analyses in the field of open source as well as proprietary ones are mentioned and the most important parts of the code for the website are described.

The resulting webpage uses Valhalla as backend and OpenStreetMap as well as Nominatim as services. Beside performing accessibility analyses it is possible to save, import, and merge results as well as to compare the reachable area to the boundaries of the alarm response area. To review the correctness of the computed isochrones they are compared with different routing services.

II Acknowledgment

At the beginning of this work I would like to say thank you to all persons that supported me during my work.

Thanks to

- the team of the Geofabrik, where I wrote my thesis for the friendly acceptance. My particular thanks to Frederik Ramm who made it possible, to work on such an interesting topic there and who ever supported me if it was necessary as well as for proofreading my report. Equally I am grateful to Rory McCann for proofreading my report several times and his help with the translation of discipline-specific words.
- my supervisors Prof. Bürg, Prof. Freckmann, and Prof. Günther-Diringer.
- Christian Stern for his programming support.
- Sabrina Futterer for proofreading my report.
- Hans-Peter Lehmann for the answers to all my silly questions and his clear explanations.

Last but not least I thank all here not namely mentioned persons that contributed to the success of this work as well as my complete studies.

III Statutory declaration

I declare that I have authored this thesis independently that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

I hereby confirm that I have written the accompanying thesis by myself, without contributions from any sources other than those cited in the text and acknowledgements.

This applies also to all graphics, drawings, maps and images included in the thesis.

.....	.....
Place and date	Signature

IV Table of contents

V List of figures

VI List of tables

VII List of abbreviations

API	Application Programming Interface
BOS	Behörden und Organisationen mit Sicherheitsaufgaben (authorities and organizations with security tasks)
BRK	Bayrisches Rotes Kreuz (Bavarian Red Cross)
CEN	Comité Européen de Normalisation (European Committee for Standardization)
CH	Contraction Hierarchies
DIN	Indicator of a German Standard, in earlier times abbreviation for Deutsche Industrie Norm (German Industry Standard)
DIN SPEC	DIN Specification, standard like specification
EMS	Emergency medical services
EN	Europäische Norm (European Standard)
ESRI	Environmental Systems Research Institute Inc.
ETOPO1	Earth Topographic database 1 arc-minute (https://www.ngdc.noaa.gov/mgg/global)
FNFW	DIN-Normenausschuss Feuerwehrwesen (DIN Standards Committee Firefighting and Fire Protection)
GeoJSON	Geographic JavaScript Object Notation
GIS	Geographic Information System
GMTED	Global Multi-resolution Terrain Elevation Data 2010 (https://topotools.cr.usgs.gov/gmted_viewer)
GUI	Graphical User Interface
JSON	JavaScript Object Notation
JSTS	JavaScript Topology Suite
LOD	Level of detail
NED	National Elevation Dataset (https://nationalmap.gov/elevation.html)
ODbL	Open Data Commons Open Database License (see https://opendatacommons.org/licenses/odbl)
OGC	Open Geospatial Consortium
ORS	Open Route Service
OSM	OpenStreetMap
OSMF	OpenStreetMap Foundation
OSRM	Open Source Routing Machine
POI	Point of interest
PPA	Personal Package Archive
RFC	Request for Comments

SPT	Shortest Path Tree
SRTM	Shuttle Radar Topography Mission (https://www2.jpl.nasa.gov/srtm)
StAN	Stärke- und Ausstattungsnachweisung (table of organization and equipment)
WGS	World Geodetic System

1 Introduction

1.1 Problem

This master thesis deals with the problem of performing an accessibility analysis for vehicles of emergency services, especially for non-police emergency services. Although there are a lot of routing services and accessibility analyses for ordinary vehicles like cars, there are none which take the special characteristics of emergency service vehicles into account.
To be able to do such an analysis the selection of a suitable software and suitable data is needed. But the suitability can only be evaluated if the requirements are known.
For this project it is desirable to use only open source software as well as open source data because they can be adapted to the personal needs and are free to use. Moreover the routing service should offer the possibility to compute isochrones. The choice of several starting points for the analyses is desirable as well as a as quick as possible computation of the result. A comprehensive, current and accurate base map and routing graph are also important requirements.

1.2 Aim

The aim of this work is the development of a website which allows users to perform an accessibility analysis for non-police emergency services. The limitation to only non-police emergency service vehicles is necessary, because they often start from a fixed location (like fire stations or ambulance stations), whereas police cars are often on patrol. An example result of a similar project can be seen in figure 26.

For this accessibility analysis OpenStreetMap data and existing open source routing software, which was developed for OpenStreetMap, should be used. Therefore their suitability has to be evaluated and the best fitting software has to be chosen.

In addition to that existing attempts to create an accessibility analysis for emergency service vehicles are pointed out.

1.3 Structure of the written thesis report

Chapter 1 gives an overview of the problem and the aim of this work as well as the structure of this report.

In chapter 2 important terms that are necessary to know later on, are defined.

The suitability of OpenStreetMap data for accessibility analysis is investigated in chapter 3.

Algorithms that are used to implement an accessibility analysis are explained in chapter 4.

The current state of the art of accessibility analysis methods is described in chapter 5.

In chapter 6 the concept of the solution developed in this work is presented. This includes a detailed target definition, the presentation of the idea of a graphical user interface and a description of the required profiles. Also a description of the tests of the available routing software for the backend and selection of routing software is located here.

Corresponding to chapter 6 the implementation of the graphical user interface, the profiles and the integration of the routing software are explained in chapter 7.

The practical testing and verification of this website is described in chapter 8.

Finally chapter 9 gives a summary of this work and an outlook for further improvements and research.

A glossary for the used emergency service specific words is attached in the appendix.

2 Definitions and explanations

2.1 Accessibility

"Accessibility is [...] a slippery notion [...] one of those common terms that everyone uses until faced with the problem of defining and measuring it"[1] This quote from Peter Gould shows how important it is to define exactly what is meant by the term accessibility in a report. The definition of accessibility depends, among other things, on the scientific discipline, for example the accessibility of a server in computer science, the accessibility of a customer hotline in customer support or the accessibility of a location in earth sciences.[2] In this paper accessibility is used in the field of earth sciences.

In this context, accessibility is the existence of a way from node A to node B within a network. The network represents, for example, the road network, the railway network or a public transport network. One or several restrictions can also be added to this like a certain travel mode (car, public transport or wheelchair), travel costs (budget, distance or travel time) or traffic regulations (one-way streets, turn or driving restrictions).[3]

Nowadays networks are often represented by graphs to be able to calculate the accessibility with a computer. An example for such a graph is the node-edge model which will be explained in the next section.

2.2 Network, Graph and Node-Edge model

According to the Duden dictionary, a network is the "Gesamtheit netzartig verbundener Leitungen, Drähte, Linien, Adern oder Ähnliches" (in English: total net-like connected pipes, cables , lines, veins or similar).[4] In the field of earth sciences networks represent for example electricity nets, the hydrographic network or the street network.[5]

A network can be represented by nodes and edges and therefore this model is often called node-edge model. Edges represents power lines, rivers or streets and always connect two nodes. An edge defines the two nodes it connects as adjacent. Nodes are endpoints of one or several edges and represent electrical substations, confluences of rivers or junctions. All edges which end in the same node are incident. In addition to that the definition of nodes is scale-dependent, too. In small-scale maps nodes describe locations like cities, in public transport networks train or bus stations and in large-scale maps crossings, road ends or general locations on which attributes are changing. This model originates from graph theory. A complete node-edge model corresponds to a finite graph.[6] "The position of the nodes, their distance from each other as well as the shape of the edges do not necessarily represent a true to scale image of the reality".[7] To adapt the graph to the reality it is possible to add the direction and different costs as attributes to the edges. In this case the graph is called a directed or weighted graph. If no attributes are added the graph is called undirected and unweighted. A directed and weighted graph is called a network. The sequence of nodes that are

[1] GOULD, 1969, p. 64
[2] SCHWARZE, 2005
[3] BARTELME, 2005, p. 124
[4] DUDEN, 2017
[5] BILL, 2010, p. 479
[6] BARTELME, 2005, p. 75
[7] BARTELME, 2005, p. 74

connected by edges is called a path. If each node of the path is just used once the path is referred to as a way. A cycle is a path, in which only the start and the end node are identical.[8]6 oben

Graphs are mainly used to do the three types of network analyses: shortest path problem, facility location problem and travelling salesman problem. An example for each of them can be found in figure 1. For reasonable network analyses directed and weighted graphs are needed. In the shortest path problem the optimal way to connect two locations is sought. This problem is often also called lowest cost problem, because the optimal way depends on the chosen type of costs like distance, time or travel costs. This is used for example for emergency service dispatching or ordinary routing.
The facility location problem seeks for the best location of a facility in respect to various factors such as accessibility to or from other places. Sample applications are a search for the best location of a projected department store, hospital or non-police emergency service station (hereinafter referred to as station) depending on several factors like travel costs, travel distance or travel time. The aim of the travelling salesman problem is the computation of the optimal route from and to a location via one or several waypoints. Therefore the starting point and endpoint have to be identical. It is used for instance to plan the travel route of postmen, suppliers or travelling salesmen.[9]

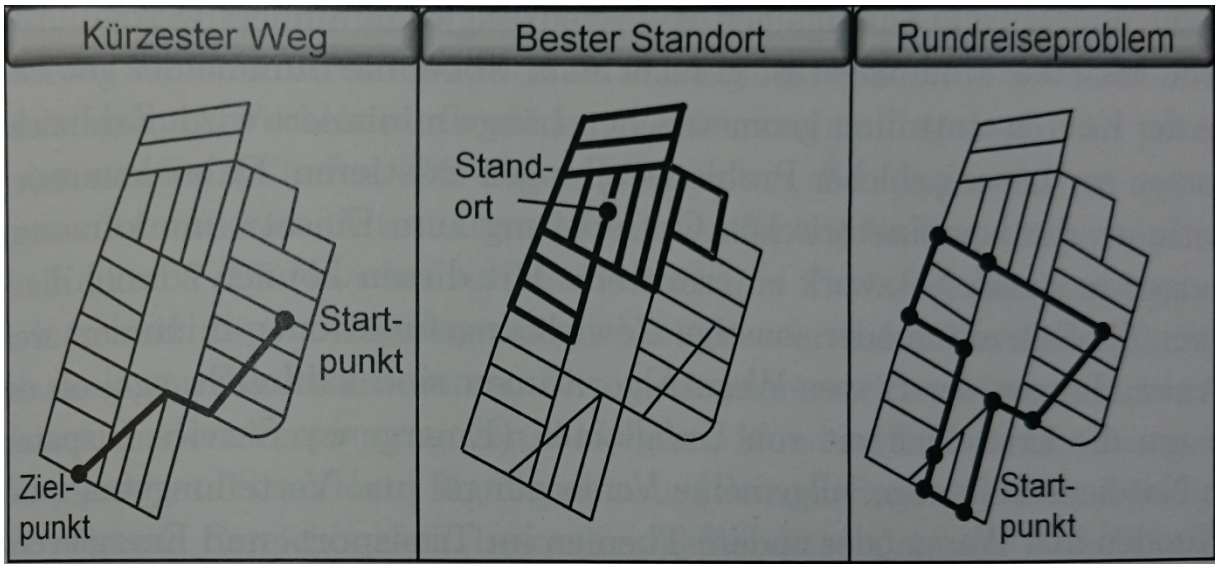

Figure 1: The three types of network analyses according to Ralf Bill: Shortest path problem, facility location problem and travelling salesman problem.[5]

2.3 Accessibility analysis

The definition of the accessibility analysis is dependent on the definition of term accessibility. Generally an accessibility analysis is a computation of the accessibility within the graph. In this case accessibility should be measured as the time that is needed to reach an area from a starting point. Therefore accessibility analyses are assigned to the group of facility location problems.[10]
It is also possible to compute the reachable region originating from several starting points. The graph represents the street network. Therefore a directed graph is used for the calculations. Hence an accessibility analysis comprises the investigation of the accessibility of a region from one starting

[8] BARTELME, 2005, p. 75; FISCHER M. M., 2003, p. 2; SEDGEWICK, 1992, pp. 476, 481, 513-533; DOMSCHKE, 1981, p. 6
[9] BILL, 2010, pp. 479-483
[10] BILL, 2010, p. 481

point or several starting points, optional within certain costs like time, and the presentation of the result on a map.

A simple accessibility analysis with only two cases is the analysis whether a location is reachable from others or not (unweighted graph). Figure 2 illustrates such an example. It is possible to reach the locations A to D by car from each other, because there are road connections between them. But it is impossible to reach location E by car, because there is no road connection to the other locations due to the fact that E is situated on an island.

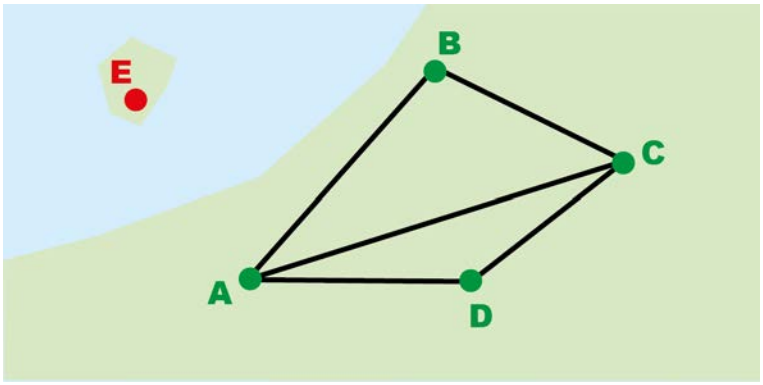

Figure 2: Result of an accessibility analysis of an undirected and unweighted graph. It is possible to reach the locations A to D from each other over the network, but it is impossible to reach location E.

More difficult, but also more significant, is the calculation and presentation if the graph is connected with different costs like distance, travel costs or time. In this case the definition of accessibility is expanded by the factor of costs: accessibility is the existence of a way from node A to node B with a limited costs (like distance, travel costs or time) within a graph. However, more detailed information about the graph is needed like length, travel costs or travel duration of all edges within the graph. For an analysis these costs are added to the edges (weighted graph).
The result of an accessibility analysis with weighted edges can be used to show which locations can be reached within a certain distance, time or with a certain budget. As shown in figure 3 the result will also vary depending on the computation method. First of all the costs (as a simple example distance) are added to the edges, this can also be done already at the construction of the graph network. Then isodistances (lines that connect all points of the same distance to the starting point) are drawn. As can be seen in the figure below there are two different methods to compute these isodistances. The left part of the figure shows the linear distance from location A, where one only has to draw circles with different radii. This is a very rough but simple approach. On the right side of the figure the computation of the distance from location A along the road axes can be seen. It is the more precise but also costlier method.

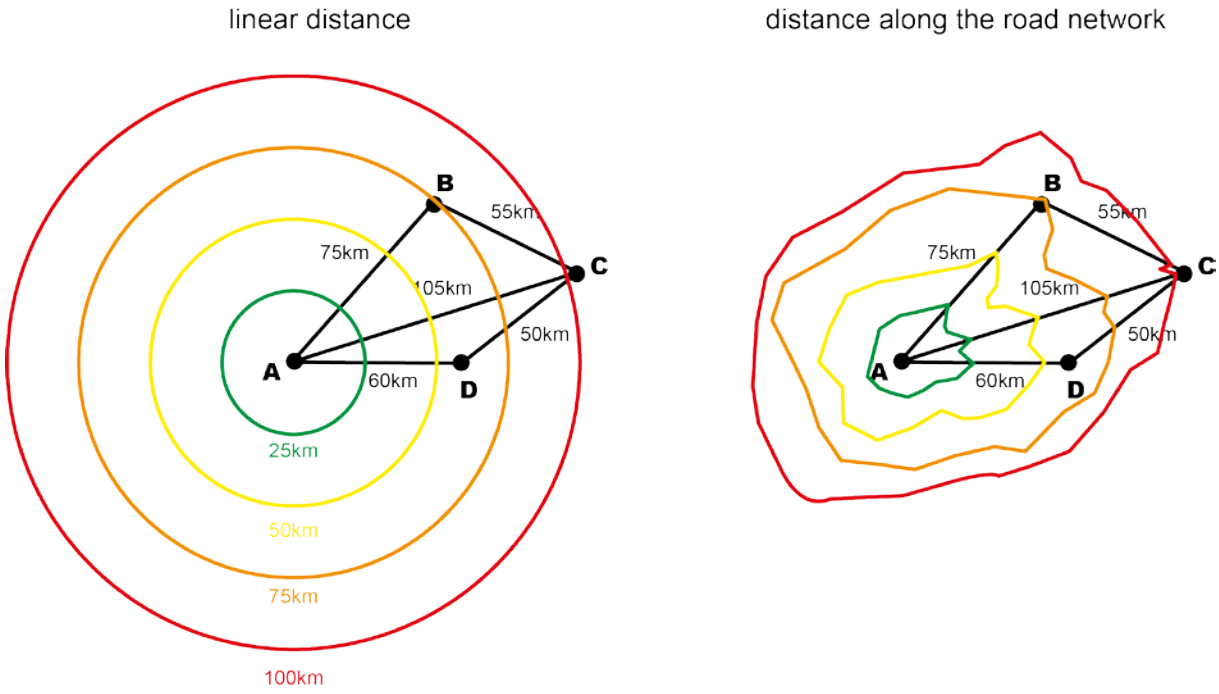

Figure 3: Result of an accessibility analysis with a directed and weighted graph. The two images show the result of two different computation methods. On the left hand side the beeline method and on the right hand side the distance along the road network.

In contrast to the distance, costs like travel time or budget have to be computed for the edges. For example the length divided by the speed yields the travel time.

An accessibility analysis can be performed by hand as well as by using a computer. The different possibilities to execute an accessibility analysis are explained in chapter 2.5.

2.4 Emergency services

Emergency services are "public organizations that respond to and deal with emergencies when they occur."[11] For different kinds of emergencies there are different types of emergency services like police, fire brigades or emergency medical services.
The regulations for response time of emergency services and the organization of emergency services themselves differ between countries as well as between federal states or provinces. Because of that the regulations are described for the German state of Baden-Württemberg.

In Baden-Württemberg, and also in Germany generally emergency services are called Authorities and Organizations with Security Tasks ("Behörden und Organisationen mit Sicherheitsaufgaben" – BOS). They are divided into police BOS and non-police BOS or into governmental BOS and non-governmental BOS[12], but police and governmental BOS, and non-police and non-governmental BOS, do not contain the same emergency services. Police BOS cover all kinds of police, the protection of the constitution and customs; non-police BOS include amongst others all kinds of fire brigades, emergency medical services, rescue helicopters, the Federal Agency for Technical Relief ("Technisches Hilfswerk" – THW), the Federal Office of Civil Protection and Disaster Assistance ("Bundesamt für Bevölkerungsschutz und Katastrophenhilfe" – BBK) and civil protection.
Governmental BOS contain all BOS which are run by the government, these are all police BOS but

[11] OXFORD DICTIONARY, 2017
[12] DEUTSCHES BUNDESAMT FÜR BEVÖLKERUNGSSCHUTZ UND KATASTROPHENHILFE, 2005

also the Federal Agency for Technical Relief and the Federal Office of Civil Protection and Disaster Assistance.
In contrast to emergency services like police, the Protection of the Constitution and ambulances most members of THW (99% of all THW members) and fire brigades (96 % of all fire fighters) are volunteers, which means they have to drive to the THW or fire station in case of an emergency before they can drive with an emergency vehicle to the scene.[13] Career fire fighters and members of the emergency medical services (EMS) are already at their station. This influences the maximum possible travel time of the emergency services which is explained more detailed in chapter 2.5.

2.5 Response time and its separate phases

In the field of emergency services the travel time is one part of the response time, which consists of the three phases talk and disposition time, turnout time and travel time.

Reporting time, however it is not part of the response time, is linked to response time because it is not predictable. Reporting time is defined as the time interval which starts with the incident (e.g. the outbreak of a fire) and ends when the incident is reported. Talk and disposition time contains the time interval starting from the begin of the emergency call until the alerting of the emergency services. The time interval between alerting and turnout of the emergency services is called turnout time. Finally the last phase of the response time is called travel time and is defined as the duration of travelling from the station to the incident. Often there is also a differentiation between response time 1 and response time 2. In this case response time 1 defines the response time for the first unit and response time 2 the response time for succeeding units.[14] The duration of each phase varies between different emergency services and defining organizations. An overview of the different phases of the response time can be seen in figure 4.

Based on the results of a German study called "Entwicklung eines Systems zur Optimierten Rettung, Brandbekämpfung mit Integrierter Technischen Hilfeleistung" (Development of a system to optimize rescue, fire fighting with the use of integrated technical assistance – ORBIT) in 1978 organizations like the Arbeitsgemeinschaft der Leiter der Berufsfeuerwehren (Working Group of the Chiefs of Career Fire Brigades – AGBF) or Landesfeuerwehrverbände (State Fire Brigade Associations – LFV) defined the maximum durations of the phases in case of a fire. This study was done by Porsche AG and WIBERA Wirtschaftsberatung AG on behalf of the German Federal Ministry of Education and Research to identify how long a human being is able to survive in a fire. For that the standard incident "critical apartment fire in an upper floor" was analyzed. The result of this study was that the time limit for start of a successful reanimation of a person is 17 minutes after the start of a fire.[15] Some experts and other studies questioned the results of the ORBIT study and also the response times derived from it, because it takes almost only career fire brigades into account, building materials and construction methods have changed and the definition of reporting time and the time for rescue operations are set arbitrarily. Moreover a new study discovered that they are too short.[16] However the results are still used to define the response time as a planning and quality characteristic in Germany today (e.g. References to the capability of the fire brigades in Baden-Württemberg).[17]

[13] DEUTSCHES BUNDESMINISTERIUM DES INNERN, 2017; THW
[14] LANDESFEUERWEHRVERBAND BADEN-WÜRTTEMBERG, 2008, pp. 15-18
[15] PORSCHE AG, 1978, pp. 23-24
[16] LINDEMANN, 2011; FISCHER C. , 2013; WIKIPEDIA: HILFSFRIST, 2017
[17] LANDESFEUERWEHRVERBAND BADEN-WÜRTTEMBERG, 2008, p. 7

The State Fire Brigade Association of Baden-Württemberg defines the time interval for the reporting time 1 in case of a fire as 2 minutes, further 2 minutes are estimated for talk and disposition time, and for turnout time and travel together time 10 minutes are scheduled. The sum of all these time intervals is 14 minutes. This means 3 minutes are left for search and rescue of people to finally meet the 17 minutes mentioned previously.[17] But these 3 minutes for search and rescue, also called time for investigation and time until the first actions become effective, are not part of the response time, because the response time ends with arrival at the scene. Five minutes after the first unit succeeding units have to arrive at the scene.
As already mentioned, the times for each of the phases differ between different regulations and guidelines. For example the AGBF calculates 3.5 minutes for the reporting time 1, 1.5 minutes for the talk and disposition, and 8 minutes for turnout time and travel time together. In this case 4 minutes are remaining for search and rescue of people. Succeeding units have to be at the scene no later than five minutes after the first unit.[18] The different phases of the response time and their duration according to the AGBF guideline can be seen on the timeline in figure 4.

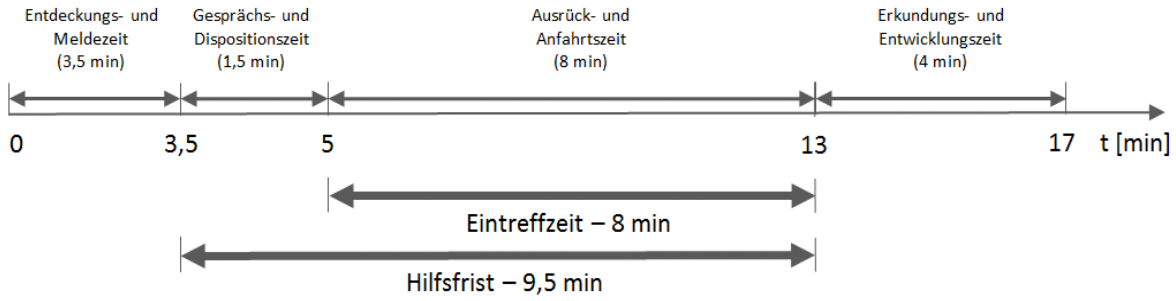

Figure 4: Phases of an emergency: reporting time, talk and disposition time, turnout and travel time and the time for investigation and time until the first actions become effective. Only the phases talk and disposition time, turnout and travel time belong to the response time.[19]

Moreover the combined time for turnout and travel can be split into 1.5 or 4 minutes for turnout time (depending on career fire fighter or members of emergency medical services, which are already at the fire station, and volunteer fire fighters, which have to drive to the fire station first in case of an emergency) and 6.5 or 4 minutes for travel time (in each case 8 minutes minus 1.5 minute respective 4 minutes).[20]

There are also response times for technical assistance, which are oriented towards the response time for EMS. Both, an ambulance and the first unit in case of technical assistance should be at the scene at least 10 minutes after the alarm. Succeeding units have to arrive at most five minutes later. For special vehicles own response times are defined, depending on the type of vehicle.[21]

In addition to the response time there are also guidelines about functional strength at the scene and degree of achievement of the response time.[22]
With the help of an accessibility analysis it is possible to determine the region which is reachable

[18] ARBEITSGEMEINSCHAFT DER LEITER DER BERUFSFEUERWEHREN, 2015, p. 3
[19] PREVENT ING, 2017
[20] PLATTNER, 2003; STADT KARLSRUHE BRANDDIREKTION, 2006, p. 42
[21] LANDESFEUERWEHRVERBAND BADEN-WÜRTTEMBERG, 2008, pp. 17, 22-24
[22] ARBEITSGEMEINSCHAFT DER LEITER DER BERUFSFEUERWEHREN, 2015, p. 1

within the required travel time from one or several stations, if influences of traffic situation and weather conditions are neglected.

2.6 Isochrones

The word isochrone is derived from the Greek iso (ἴσο) = equal and chronos (χρόνος) = time. This means isochrones are isolines which connect all points that can be reached within a certain time interval from a defined central point or from which the defined point can be reached in a certain time interval. It is also possible to define several central points however the determination of the travelling duration to a specific central point can be difficult or impossible then. Isochrone maps with one central point are called monocentric isochrone maps, with several starting points they are called polycentric isochrone maps. The extent of the isochrones differ in dependency on the used mode of transportation (airplane, public transport, car, truck, bicycle, pedestrian, ...) and the computation method (shortest way, fastest route, assumption of no traffic, current traffic conditions, ...).[23] But "all travel time figures are at best approximate of course. In practice they will vary with conditions of the route, vehicle, weather, the time of day, and so on."[24] Beside isochrones the area reachable within a defined time interval can also be shown by the use of polygons to make the map easier to read. Hake, Grünreich and Meng recommend to use at most 8 to 10 different time intervals in one map.[25]

Based on a transportation network like OSM a shortest path tree (SPT) has to be computed to get the travel time from a starting point to all surrounding points. Therefore algorithms like Dijkstra or A* are used.[26] Isochrones are the result of an interpolation between measurement points. There are different interpolation methods, some of them are listed on the OSM Wiki webpage for isochrones (http://wiki.openstreetmap.org/wiki/Isochrone).
Generalization of isochrones can be done through selection of individual isochrones (increase of time interval between two isochrones) or simplification of the isochrones themselves. While doing that, care has to be taken to retain or stress important characteristics like influence of the relief or road network.[27]

2.7 Open source

Open source is a model for developing and offering not just free but also unrestricted products of different kinds such as data, knowledge or software. This includes the abstinence of "restrictions on copying, redistribution, understanding and modification."[28] Open source software is the most common known open source product, but it is also used in many other fields. Several of them can be seen in figure 5. Examples for open source products are Linux, OpenStreetMap and Wikipedia.

[23] WITT, 1970, pp. 344-346; GORKI & PAPE, 1987, pp. 322-324
[24] MUEHRCKE, 1980
[25] HAKE, GRÜNREICH, & MENG, 2002
[26] OPENSTREETMAP WIKI: ISOCHRONE, 2017
[27] WITT, 1970, pp. 344-346
[28] STALLMAN, 1986, p. 8

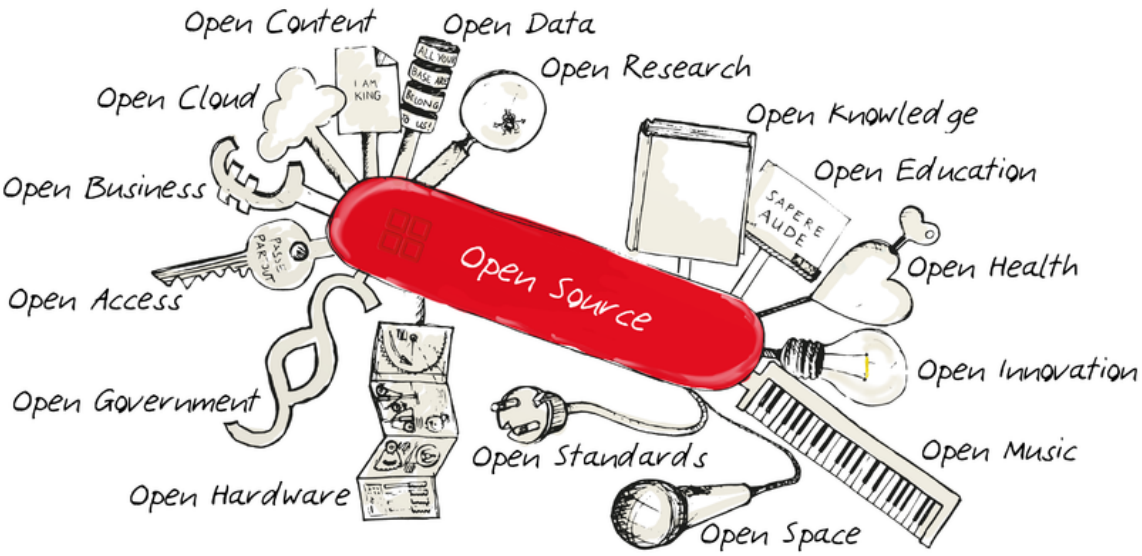

Figure 5: Various kinds of Open source.[29]

There are subtle differences between open source and free software, but these are not relevant for this work.[30]

2.8 OpenStreetMap

OpenStreetMap is a project to collect free geodata and create a free and open source map of the world. It was founded in 2004 by Steve Coast in the United Kingdom.

The OSM project is comparable to Wikipedia. Everybody can contribute and there are no rules how something has to be drawn and in which level of detail (LOD). However a registration is necessary for mapping (adding something to the OSM data) and in the course of time also some arrangements have evolved for structuring the map objects. New data is added through GPS tracking, mapping on the basis of satellite images released for this purpose, local knowledge or import of geodata released for this purpose (like Corine Land Cover or TIGER data).[31]

Figure 6: Logo of OpenStreetMap.

OSM is still under active development. The number of contributors (called mapper in OSM) and their additions to the map respectively their changes on the map has been increasing since the start of the project.

OpenStreetMap uses a special data model to store the raw data in a common database and for an OpenStreetMap specific type of XML data format, the OSM-XML data format.

This data model offers the three different object types node, way and relation. An anomaly of this data model is the absence of a polygon object type. Instead of polygons the OSM data model uses closed ways.
Nodes are used to display Points of interest (POIs) or other objects that can be represented by a

[29] WIKIPEDIA: OPEN SOURCE, 2017
[30] SYDEKUM, 2013
[31] RAMM & TOPF, OpenStreetMap - Die freie Weltkarte nutzen und mitgestalten, 2010, pp. 3-9

point. Moreover they can be connected through ways and therefore be part of linear or areal objects like streets or lakes. Ways represent linear objects like fences, rivers or streets or polygons. Relations offer the possibility to add connections between different objects, for instance to define several ways as part of a public transport route, river or street.
For precise description of an object various attributes can be added. These attributes are called tags. Each tag consists of a key and a value, often written in the form key = value. Figure 7 shows an example of several tags in Ettlingen, Germany. The number of tags for one object is not limited. As mentioned before polygons are modeled by closed ways with area-specific tags. Each object (no matter whether node, way or relation) is identifiable by a unique identifier (ID). Apart from this ID each object contains an arbitrary number of attributes, administrative information (last change time, user name, version, ...) and additional object specific information. In case of nodes these specific information are their coordinates in the World Geodetic System 1984 (latitude and longitude). For ways these specific information means an ordered list of the connected nodes and for relations an ordered list of members (nodes, ways and relations) and their role within the relation.[32]

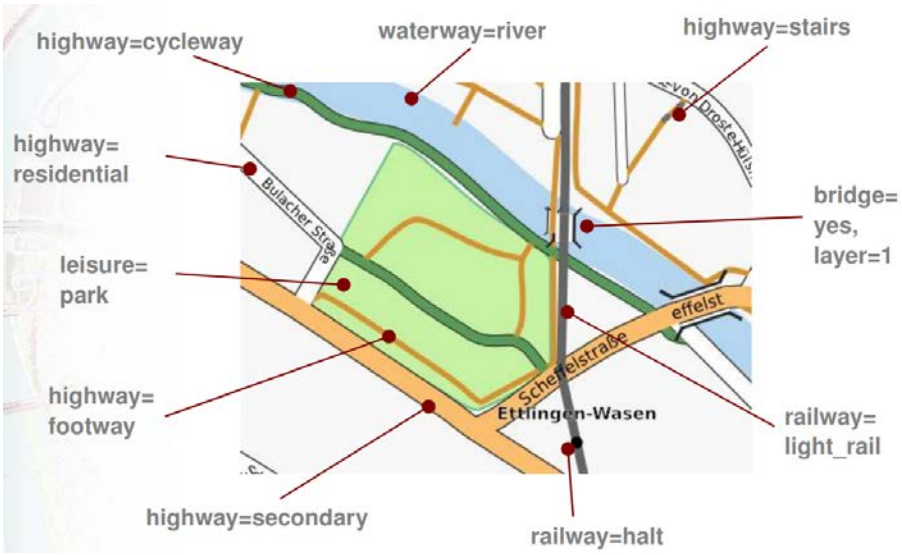

Figure 7: Example of several key-value pairs used in OpenStreetMap.[33]

Out of this raw data maps can be rendered. On the official website four different map styles can be found, but none of them displays all of the information that OSM offers. It is also possible to download the raw data to use it for one's own maps.[31] There are already various mashups that use OSM data for example to display a map or navigate. A comprehensive list of them can be found at the OSM Wiki (https://wiki.openstreetmap.org/wiki/List_of_OSM-based_services).

The OSM-XML data format is used to exchange OSM data and has the extension .osm. Figure 8 shows an example of an OSM-XML file. The `<osm></osm>` element covers the complete content consisting of an optional bounding box element `<bounds />` and a list of OSM objects. The OSM objects are ordered according to the object type; first the nodes, then the ways and finally the relations.[32]

OSM uses the Open Data Commons Open Database License (ODbL) for its data. This means data have to be shared with the same rights as the original data had, moreover OSM has to be mentioned as

[32] RAMM & TOPF, OpenStreetMap - Die freie Weltkarte nutzen und mitgestalten, 2010, pp. 55-61
[33] RAMM, OpenStreetMap Die freie WikiWeltkarte Übersicht – Chancen – Grenzen, 2011, p. 29

the source. The exact license can be found at the webpage https://opendatacommons.org/licenses/odbl.[34]
In contrast to other free-of-charge map providers, like GoogleMaps it is possible to get and use the raw data as well as the maps without paying any fee or asking for permission.

The OpenStreetMap Foundation (OSMF) supports and represents the project for example in license questions. OSMF also takes care of the servers and hosts the official OSM website www.openstreetmap.com.

```
<osm version="0.6" generator="OpenStreetMap server">
    <bounds minlat="48.970872" minlon="8.300709" maxlat="49.019140" maxlon"8.456753"/>

    <node id="28422710" timestamp=2007-05-02 02:45:02" user="SlowRider
          uid="99123" version="2" changeset="1234" visible="true"
          lat="48.9751327" lon="8.3104325">
        <tag k="amenity" v="bank"/>
        <tag k="name" v= "Volksbank"/>
        <tag k="addr:housenumber" v="23"/>
    </node>

    <way id="4060164" timestamp="2006-12-08T11:58:49+01:00" user="Jochen Topf"
          uid="9999" version="1" changeset="1234" visible="true">
        <nd ref="1637589">
        <nd ref="21498581">
        <nd ref="21498576">
        <nd ref="14911465">
        <nd ref="11238271">
        <nd ref="11238273">
        <nd ref="11238274">
        <tag k="name" v="Gerwigstraße"/>
        <tag k="highway" v= "residential"/>
        <tag k="maxspeed" v="30"/>
    </way>

    <relation id="2400" timestamp="2007-10-17T10:17:04+01:00"
          user="Franc" uid="9922" version="13" changeset="3752">
        <member type="node" ref="15168225" role="via"/>
        <member type="node" ref="2779788" role="to"/>
        <member type="node" ref="3169554" role="from"/>
        <tag k="restriction" v="no_right_turn"/>
        <tag k="type" v= "restriction"/>
    </relation>
</osm>
```

Figure 8: Example of an OSM-XML file which contains a node, a way and a relation.

2.9 GeoJSON

"GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON). It defines several types of JSON objects and the manner in which they are combined to represent data about geographic features, their properties, and their spatial extents."[35] GeoJSON files use the file extension geojson and are used to present geodata in web applications.
A first specification of GeoJSON was published in 2008. In August 2016 this specification was replaced

[34] OPENSTREETMAP: COPYRIGHT AND LICENSE
[35] BUTLER, DALY, DOYLE, GILLIES, HAGEN, & SCHAUB, 2016, p. 1

by the standard specification RFC 7946 developed by the GeoJSON format working group. The GeoJSON standard is an open standard format.

It is possible to store the geometry types Point, LineString (Line), Polygon, MultiPoint, MultiLineString (Line), MultiPolygon and GeometryCollection in GeoJSON.[36] In the RFC 7946 the geometry types are divided into "single type Geometry objects (Point, LineString, and Polygon) and homogeneously typed multipart Geometry objects (MultiPoint, MultiLineString, and MultiPolygon)" as well as heterogeneously typed multipart Geometry objects (GeometryCollection). Whereas single type Geometry objects and homogeneously typed multipart Geometry objects can only contain geometries of the same geometry type, it is possible to union different geometry types into a GeometryCollection.[37] Additional information to the geometry objects can also be stored. This additional information is called properties. Geometry object and properties together are combined to so-called Feature objects. A FeatureCollection includes several Feature objects. The geometry is saved in the World Geodetic System 1984 (WGS84) and decimal degree notation.

Figure 9 shows an example of a GeoJSON file which contains a Point, a LineString and a Polygon geometry each time with associated properties in three different Feature objects. All Feature objects are contained in a FeatureCollection.[38] The standard also defines some additional attributes like identifier (`"id": "f1"`) and bounding box (`"bbox": [100.0, 0.0, 105.0, 1.0]`). In addition to that it is also possible to define custom attributes (so-called foreign members), but these reduce the interoperability of the GeoJSON files.

The coordinates attribute consists out of two or three elements; longitude, latitude and optionally the elevation relating to the WGS84 reference ellipsoid in meters. There is no limit of decimals of the coordinates, but the more precise the coordinates the larger the file size. Normally six decimals are enough. This corresponds to about 10 centimeters.

Polygons can have holes. In this case the first sequence of coordinates defines the exterior line and all other coordinates sets define the interior lines (holes).[39]

[36] GEOJSON WORKING GROUP; BUTLER, DALY, DOYLE, GILLIES, HAGEN, & SCHAUB, 2016, pp. 3, 4
[37] BUTLER, DALY, DOYLE, GILLIES, HAGEN, & SCHAUB, 2016, p. 9
[38] BUTLER, DALY, DOYLE, GILLIES, HAGEN, & SCHAUB, 2016, pp. 1, 3-6
[39] BUTLER, DALY, DOYLE, GILLIES, HAGEN, & SCHAUB, 2016, pp. 9, 12, 18

```
{
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "geometry": {
            "type": "Point",
            "coordinates": [102.0, 0.5]
        },
        "properties": {
            "prop0": "value0"
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [102.0, 0.0],
                [103.0, 1.0],
                [104.0, 0.0],
                [105.0, 1.0]
            ]
        },
        "properties": {
            "prop0": "value0",
            "prop1": 0.0
        }
    }, {
        "type": "Feature",
        "geometry": {
            "type": "Polygon",
            "coordinates": [
                [
                    [100.0, 0.0],
                    [101.0, 0.0],
                    [101.0, 1.0],
                    [100.0, 1.0],
                    [100.0, 0.0]
                ]
            ]
        },
        "properties": {
            "prop0": "value0",
            "prop1": {
                "this": "that"
            }
        }
    }]
}
```

Figure 9: Code example of a GeoJSON file which contains a Point, a LineString and a Polygon geometry.[38]

3 The suitability of OpenStreetMap data for accessibility analysis

OpenStreetMap offers free geodata for the whole world. The quality of the data varies, depending among other factors on the location in the world. In 2010 Frederik Ramm and Jochen Topf determined in their book about OpenStreetMap that many cities worldwide are mapped in more detail in OSM than in maps of Google, Yahoo or Microsoft and are more up to date than printed city maps. Also OSM offers better geodata for developing countries than anybody else is able to provide.[40] But on the other hand especially in rural regions the OSM data has to be improved. The German OpenStreetMap community came to the same conclusion.[41]

In order to evaluate OSM for accessibility analyses, it is required to define which features are necessary to perform this kind of calculation.
An exhaustive road network is of primary importance for an accessibility analysis. This network should contain information about speed limits, turn restrictions and vehicle dimension limits as well as vehicle weight restrictions. Moreover house polygons or at least polygons for built-up areas are necessary for interpreting the result and to check if all houses are reachable in the mandatory response time. In addition to that the borders of districts, municipalities and counties are required to inspect the coverage of the alarm response areas of each station. Knowing the locations of existing stations is also beneficial, but these should be known by the local responsible persons that perform the accessibility analysis. Obviously currency and accuracy of the information are as important as the completeness.

Accessibility analysis needs large-scale maps with a map scale of about 1:50,000. This correspondents to zoom level 14 (1:35,000) in online maps like OpenStreetMap, Google Maps or Here Maps.[42] To evaluate the quality of OpenStreetMap, its data it is compared with other online map providers. This can easily be done by the use of the map compare tool of Geofabrik (http://tools.geofabrik.de/mc). With this tool it is able to compare up to 8 maps or satellite images to each other. Some of the selectable options are just different map styles of the same data basis. Therefore OpenStreetMap is only compared with Google Maps and HERE maps in this work. Also two different places are chosen to do so. On the one hand the German city of Karlsruhe and on the other hand Karamay in China.

Karlsruhe is a city with around 300,000 inhabitants which is mapped very extensively.[43] A city with a comparable number of inhabitants in China is Karamay City in the North East of China close to Ürümqi. Karamay City has more than 400,000 inhabitants, but because it also has an area of over 7,700 km² only the district Karamay (circa 260.000 inhabitants) is chosen to make a comparison.[44] OSM depends highly on volunteers to map their surrounding area. Because freedom in China is limited mapping is made difficult. Therefore it can be assumed that this city is not well mapped.

[40] RAMM & TOPF, OpenStreetMap - Die freie Weltkarte nutzen und mitgestalten, 2010, pp. 3, Color Plate 13
[41] OPENSTREETMAP GERMANY
[42] OPENSTREETMAP WIKI: ZOOM LEVELS, 2016
[43] DESTATIS, 2016
[44] TJCN, 2016

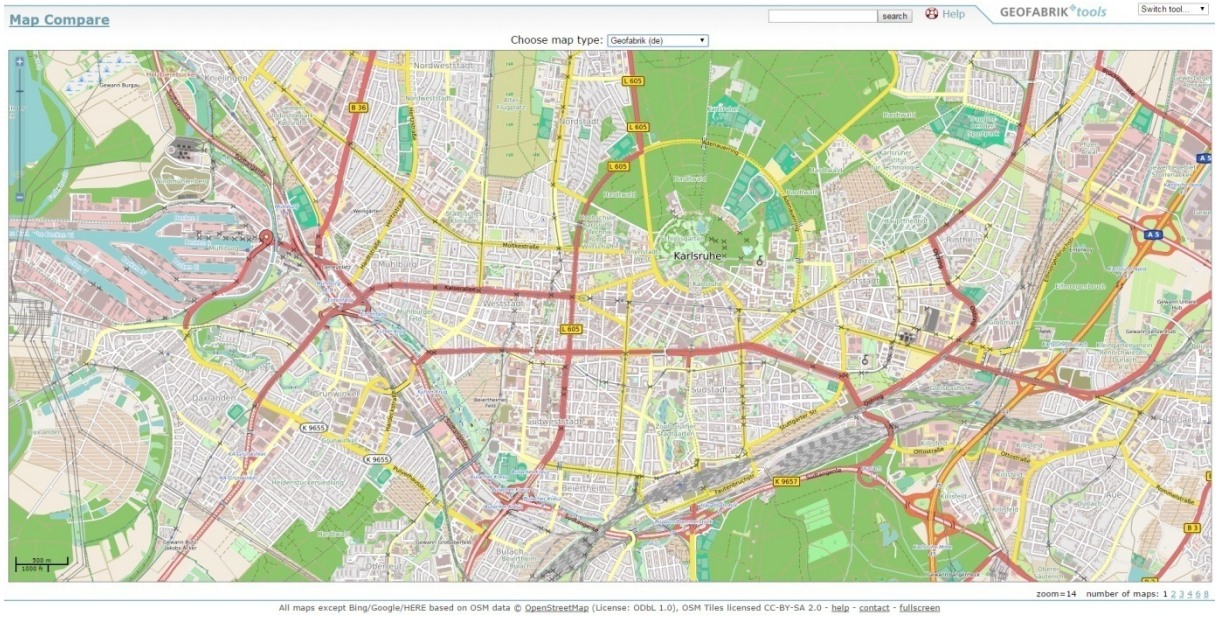

Figure 10: Map section of Karlsruhe of OpenStreetMap in zoom level 14.[45]

Figure 10 shows a map section of OpenStreetMap at the end of march 2017 of Karlsruhe in zoom level 14. As can be seen the road network is very dense. Also small ways in forests or parks are mapped together with details about the road category. Moreover the house polygons are included and a few street names are visible. By increasing the zoom level more street names, points of interest (POIs) and house numbers become visible.

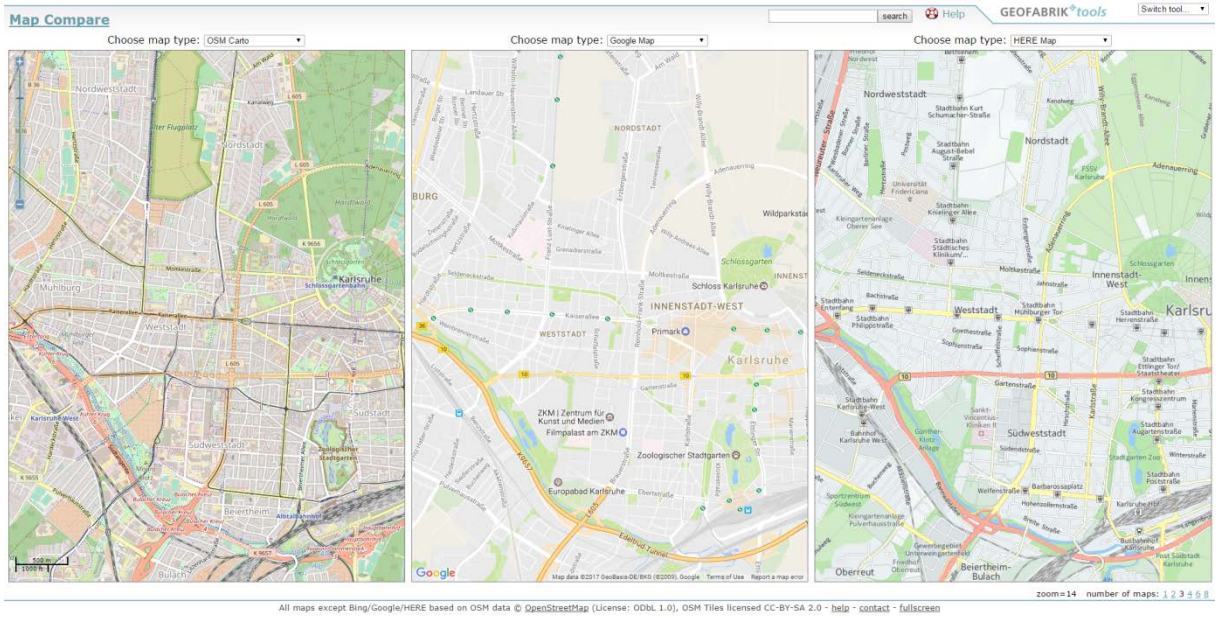

Figure 11: Map comparison of Karlsruhe in zoom level 14. The left map is OpenStreetMap, the map in the middle is Google Maps and the right map is HERE Maps.[45]

A map comparison of OpenStreetMap with maps of Google and HERE can be seen in figure 11. The result of the comparison is that the quality of the OpenStreetMap map of Karlsruhe is comparably good as the map of Google Maps and significantly better than the HERE map.
There are far more details like house polygons or sports grounds than in the maps of Google and

[45] GEOFABRIK, 2016

HERE. The path network is more comprehensive. Altogether the OpenStreetMap map section shows the most information of the three map sections. But Google Maps and HERE Maps showing more street names at this zoom level.

The quality of the map section of Karamay in contrast to the one of Karlsruhe is very poor. Only the main roads and a few minor roads are present in the OpenStreetMap map as figure 12 shows. There are just a few house polygons in the center of Karamay. After increasing the map scale only few additional information appears.

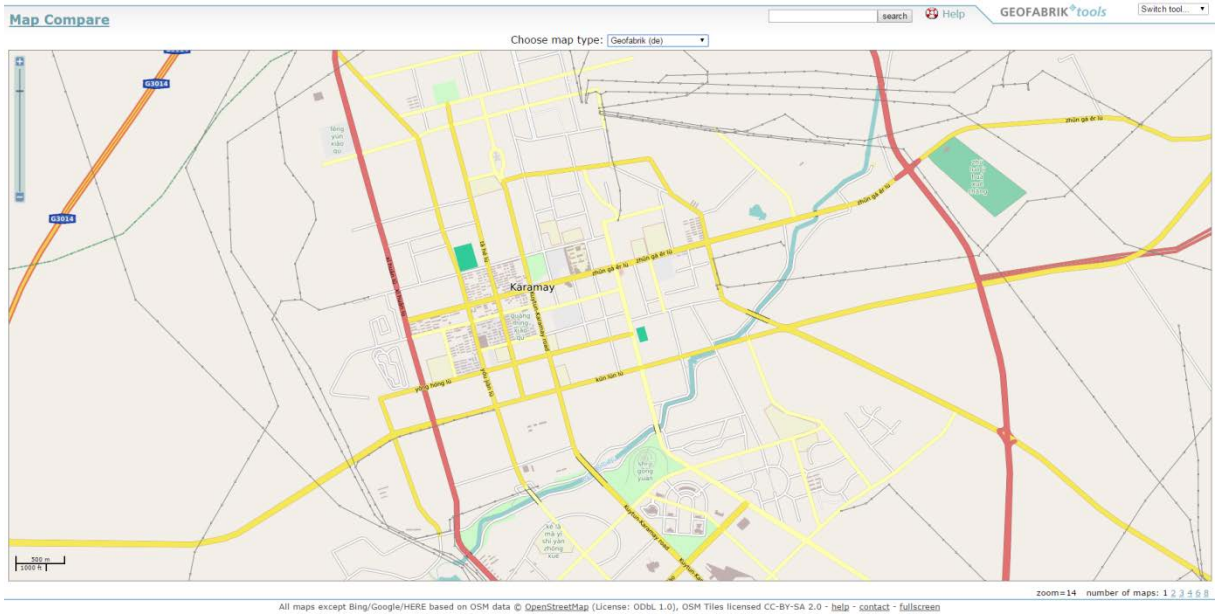

Figure 12: Map section of Karamay of OpenStreetMap in zoom level 14.[45]

A comparison of the map of OpenStreetMap with Google Maps and HERE Maps is visible in figure 13. The quality of Google Maps is significantly better, because it contains not only more roads but also more points of interest. In contrast to that the map of HERE contains nearly no information.

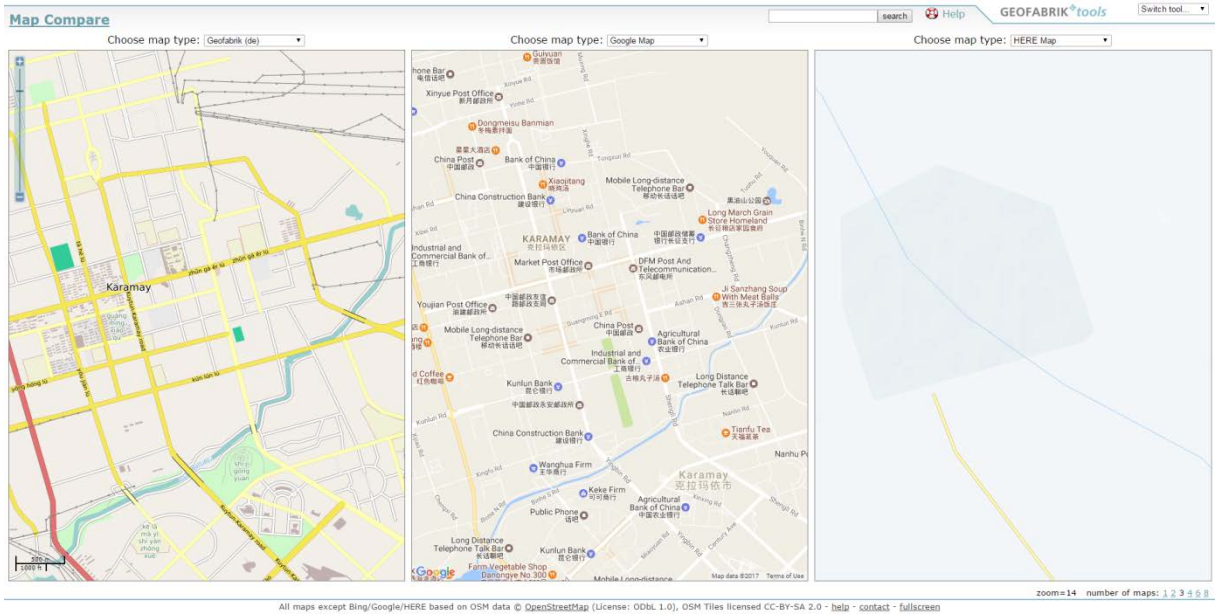

Figure 13: Map comparison of Karamay in zoom level 14.[45]

As these two examples show, OpenStreetMap data is comparable to Google Maps and better than HERE Maps. In some parts the quality is even better than maps of Google. But the quality of OpenStreetMap is not consistent in all areas, sometimes the map is only rudimentary as the example of Karamay shows. In this investigated case Google Maps offers more information, but the OSM map quality is by far better than the HERE map. It has to be noted that not all available information is displayed in the maps at all or in individual zoom levels.

OpenStreetMap has the capability to be more up to date than other map providers, but the currency depends on the volunteers. Everybody can update the OSM data, changes are already visible a few minutes after uploading them. New roads or houses as well as closed roads are added to OpenStreetMap faster than to the others, because the number of contributors is higher. Although the risk of misuse is lager if everybody is allowed to change whatever they want, errors are corrected faster. Users can be blocked if they deliberately add wrong information to OSM several times after they have been cautioned. The knowledge about the local conditions of the local mapper provides additional important information.
Another advantage of using OpenStreetMap data is that if the available data for the desired area is not good enough the user is able to improve it. Therefore the quality of OpenStreetMap is increasing every day. Contributing is not that easy for Google Maps or HERE Maps.

Comparisons indeed are only suitable to get a first overview of the suitability of OSM data, because even Google Maps or HERE Maps do not contain all information worldwide. Therefore the fulfillment of the requirements earlier defined in this chapter are reviewed in more detail.

A recently published study of Christopher Barrington-Leigh and Adam Millard-Ball investigated the completeness of OSM road network with the two different methods visual assessment of 45 points per country and saturation of contributions based on the roads already mapped in OSM itself.[46] They identified that the road network is "globally around 83% complete, and more than 40% of countries - including several in the developing world - have a fully mapped street network"[47] in January 2016 by using the visual assessment method[48] and 97% by using the saturation method. [49] In addition to that 77 of the 185 investigated countries "are more than 95% complete."[49] But countries like "Kiribati, Afghanistan, Egypt and China are all less than one-third complete."[50] The completeness of the road network in OSM per country is visible at figure 14.

[46] BARRINGTON-LEIGH & MILLARD-BALL, 2017, pp. 4-8
[47] BARRINGTON-LEIGH & MILLARD-BALL, 2017, p. 1
[48] BARRINGTON-LEIGH & MILLARD-BALL, 2017, p. 9
[49] BARRINGTON-LEIGH & MILLARD-BALL, 2017, p. 10
[50] BARRINGTON-LEIGH & MILLARD-BALL, 2017, p. 11

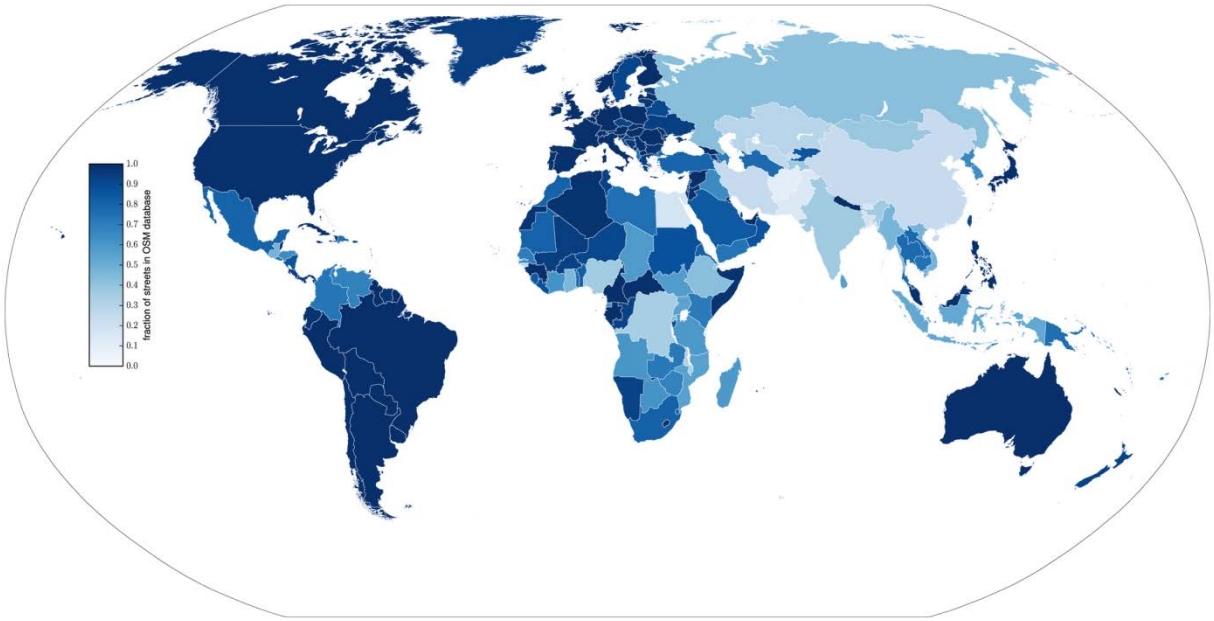

Figure 14: Completeness of the OpenStreetMap road network per country in January 2016.[51]

Information about speed limits, turn restrictions and vehicle dimension limits as well as vehicle weight restrictions are not completely added to OSM at the moment. However a non existing restriction tag not always indicates missing information, because if no tag is added the default values of OSM for the particular road type is used (in fact in this case for example `maxspeed=none` should be used). Figure 15 shows a screenshot from http://product.itoworld.com/map/124 where all tagged speed limits worldwide can be seen. At http://maxheight.bplaced.net/overpass/map.html a map that visualizes where vehicle dimensions are defined and where they are missing (only height restrictions) can be found.

The completeness of speed limits, turn restrictions, and weight restrictions is not mandatory, because emergency vehicles are exempt of speed limits and turn restrictions as long as no other road user is endangered or forced. Emergency vehicles are also often exempt of weight restrictions if the weight restrictions are declared to protect a building that will be damaged by frequent use of too heavy vehicles. On the contrary restrictions of the vehicle dimensions cannot be ignored, because narrow passages are impassable with larger dimensions.[52]

At the moment there are no studies that investigate this topic. It can be assumed that the completeness of vehicle dimension restrictions decrease from urban areas to rural areas, but also from the city center to the periphery. In addition to that the completeness in European countries is probably higher than for instance in African countries.

[51] BARRINGTON-LEIGH & MILLARD-BALL, 2017, p. 12
[52] HÄRDI, 2017

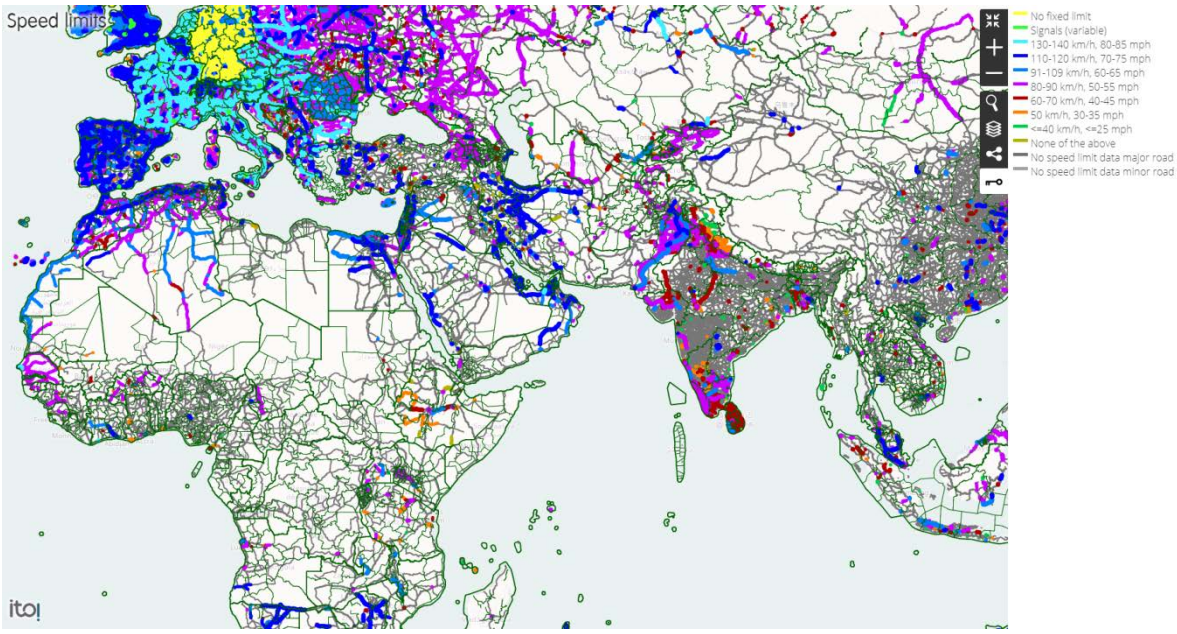

Figure 15: Set speed limit values in OSM. Clearly visible is the big difference in completeness between Europe and the other continents. [53]

Most studies that have analyzed the completeness of the data only investigated the road network. Studies which tested the completeness of house polygons compared them with official maps like ATKIS Basis-DLM, cadastral map, basemap.at, OS Street View, the topographic landscape model of swisstopo or the official vector cartography of the Milan Municipality. These comparisons indicate a much poorer completeness of house polygons in OpenStreetMap than in official maps. However the coverage is increasing. Like with all data, the completeness is high in city centers and decrease in direction periphery and rural areas. All currently available studies only investigate the completeness of house polygons limited to cities or at most one country. Some of them, and their results, are defined below.

Hahmann, Hecht, and Kunze used four different approaches to investigate the completeness of house polygons in the German states of North Rhine-Westphalia and Saxony in November 2011. In their study they detected an average coverage of circa 15% with values of the different examinations between 10.1% and 33.3%. An update of their study in the year 2012 shows an increase of the coverage to 23% and of the values for the different examinations to 14.4% respective 45.9%. This corresponds to an increase of 8% in a year. If this rate of increase will be continuous all houses will be added by 2021. In addition to that they refer to the study "OpenStreetMap in 3D – Detailed Insights on the Current Situation in Germany" of Marcus Götz and Alexander Zipf in 2013 in which it was found that 30 % of all main buildings are already mapped in OpenStreetMap and that this number is increasing by 1% each month.[54]

An Austrian study of Thomas Konrad discovered that OpenStreetMap in 2016 already contained more than 85% of all houses in Austria. In addition to that he developed an online map to compare the houses included in OSM against the houses contained in basemap.at. This map can be found at https://osm-austria-building-coverage.thomaskonrad.at/map.[55]

In 2015 the completeness of houses in the three British cities Leeds, London, and Sheffield was

[53] ITO MAP, 2017

[54] HECHT, KUNZE, & HAHMANN, Zur Vollständigkeit des Gebäudedatenbestandes von OpenStreetMap, 2014, pp. 44, 59, 74, 75, 77, 78, 80; GÖTZ & ZIPF, 2012, p. 3

[55] KONRAD, 2017

investigated by Katerina Chistopoulou, Claire Ellul and Clarire Fram. They compared the houses contained in OSM with the houses contained in OS Street View, the official geodata maintained by the British government. Thereby they discovered that the houses in Sheffield were covered to 75%, while in London only 33% and in Leeds 30% were mapped.[56]

In 2015 Lorenz Hurni, Ionut Iosifescu and Fabian Müller published a study in which they investigated the quality of buildings in OSM all over Switzerland called "Assessment and Visualization of OSM Building Footprint Quality". They discovered that OSM contains more buildings in the cantons of Basel-Stadt and Basel-Landschaft.[57] Figure 16 illustrates their results.

An Italian study of 2016 investigated the city of Milan and discovered that the official data has a 9% larger area than OSM data.[58]

In the absence of house polygons it is possible to define them as residential areas. 4,161,809 nodes, polygons, and relations are tagged as `landuse=residential`, which defines them as residential built-up areas.[59]

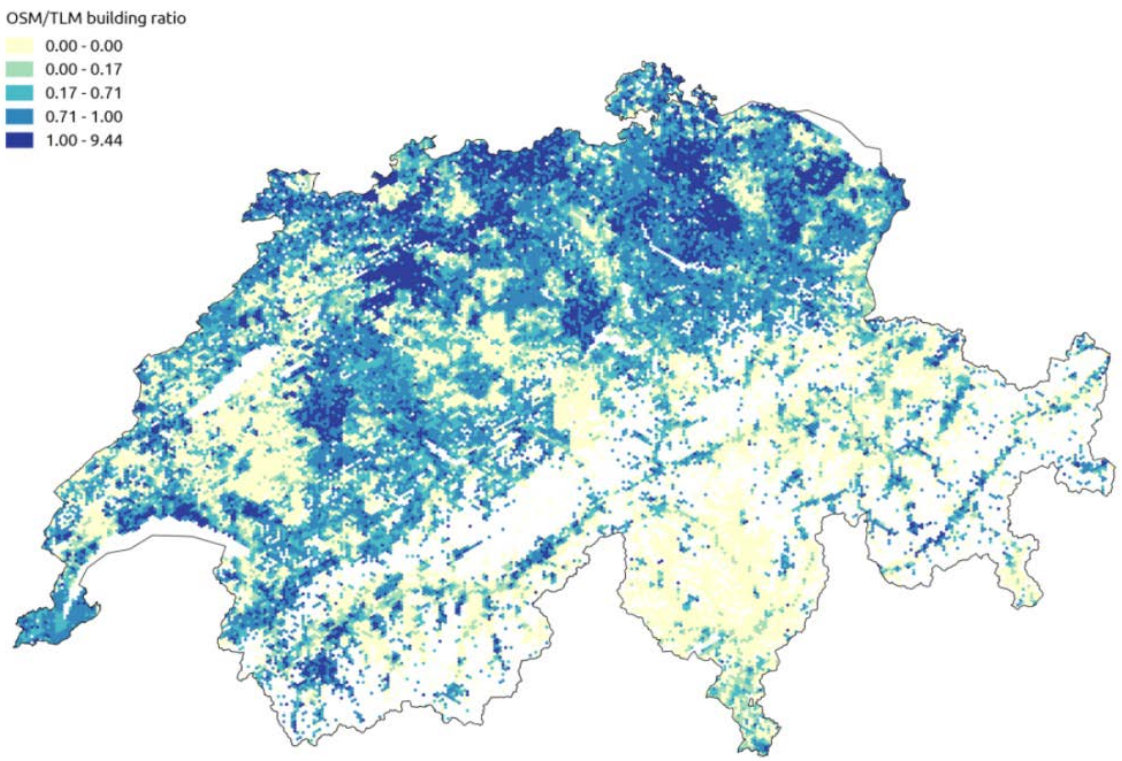

Figure 16: Completeness of OSM building data in comparison to official data in Switzerland. The darker the color the completer the buildings.[60]

OpenStreetMap contains a lot of borders on different administration levels, but not every administration level exists in each country. Not all of them are needed for investigation of the accessible region of an individual station but they are needed to make boundaries comparable between different countries.

[56] CHISTOPOULOU, ELLUL, & FRAM, 2015, p. 2
[57] HURNI, IOSIFESCU, & MÜLLER, 2015, p. 4
[58] BROVELLI, MINGHINI, MOLINARI, & ZAMBONI, 2016
[59] OPENSTREETMAP WIKI: RESIDENTIAL, 2016
[60] HURNI, IOSIFESCU, & MÜLLER, 2015, p. 7

Best fitted would be application areas like they are tagged with `fire_boundary=yes` for a special map (http://karta01.ru). Indeed this tag is not often used, only 374 objects make use of this tag.[61] Instead of application area borders also the administrative borders can be used, because they are often congruent with the application area borders are of the most importance. The administrative levels 6 and 8 which represent counties or independent cities respective cities or villages. Overall there are 1,440,343 objects tagged as boundary in OpenStreetMap. A matching of the administrative levels of OpenStreetMap to the country-specific boundary types can be found at OpenStreetMap Wiki.[62] The sources of borders are often official entities. Which administrative boundaries are already mapped in OpenStreetMap can be seen at the website of Walter Nordmann (https://wambachers-osm.website/boundaries). In most of the European countries the borders are mapped completely like for instance in Germany or Great Britain, but in other parts of the world often only borders of counties are mapped.[63] An overview of the finest grained mapped administrative boundaries level worldwide at the 8 July 2017 can be seen in figure 17. In case of missing data it can be assumed that the local authorities should be able to identify their application area.

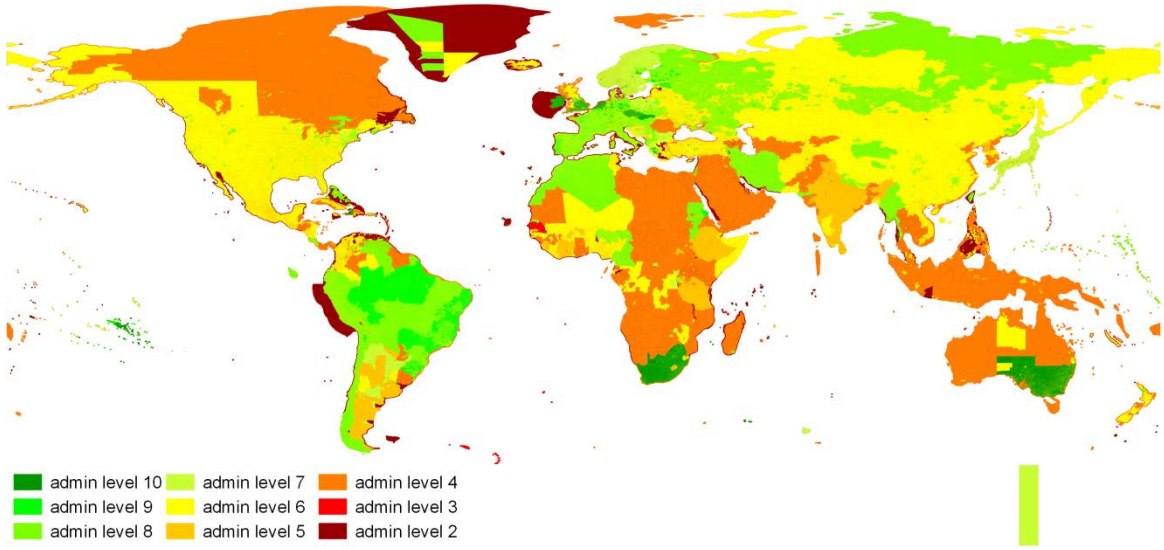

Figure 17: Finest mapped administrative boundary level worldwide at the 8 July 2017. This includes also the boundaries of the territorial waters.

The location of all fire stations added to OpenStreetMap can be seen at http://openfiremap.org. Aside from Austria , Germany, and Luxembourg, where many fire stations are mapped, information about fire stations is just in some regions (for instance Canada, Switzerland or the United States) or here and there available (for example Argentina, Australia or France). In the most countries fire stations are only mapped sparsely. Fire stations are mapped with the tag `amenity=fire_station`. At the time of writing 52,691 nodes, 37,548 polygons, and 300 relations are tagged as fire station worldwide. This would correspond to 90,539 fire stations worldwide, which is a very low value. Furthermore some fire stations are mapped as a node as well as a polygon.[64]

A special map for EMS does not exist, but there is also a statistic how often this tag is used. The tag

[61] OPENSTREETMAP WIKI: FIRE BOUNDARY, 2016
[62] OPENSTREETMAP WIKI: BORDER, 2017
[63] WIKIPEDIA: CITY BOUNDARIES OF GERMANY IN OSM, 2016; OPENSTREETMAP WIKI: CITY BOUNDARIES OF GREAT BRITAIN IN OSM, 2017
[64] OPENSTREETMAP WIKI: FIRE STATIONS, 2016

`emergency=ambulance_station` is used to tag ambulance stations. But 145 objects are tagged with `amenity=ambulance_station` instead.
Worldwide only 5,272 objects are tagged correctly as ambulance station (2,331 as node, 2,902 as polygon, and 39 as relation). Together with the wrongly tagged objects this equates to 5,417 mapped ambulance stations.[65]

A study of Mordechai Haklay, Sofia Basiouka, Vyron Antoniou, and Aamer Ather in 2010 demonstrated that the positional accuracy of OSM data increases if there are more contributors. Whereas " the first five contributors to an area seem to provide the biggest contribution in terms of positional accuracy improvement." [66] They found out that "the positional accuracy of OSM is high"[67], more than 80% of the roads mapped in OSM are at the identical position as in official data (Ordnance Survey Meridian 2).[67] Crossings are mapped with an positional accuracy below 6m if there are more than 15 contributors per square kilometer.[66] All in all "the quality of OSM data is comparable to traditional geographical datasets that are maintained by national mapping agencies and commercial providers."[68]
These results are confirmed by other studies. Ina Ludwig, Angi Voss, and Maike Krause-Traudes discovered in their study "A Comparison of the Street Networks of Navteq and OSM in Germany" that "73% of the OSM street objects in Germany are within a distance of 5m, 21% between 5m and 10m and 6% between 10m and 30m away from their Navteq partner."[69] According to Christof Amelunxen, who compared the house numbers in OSM with data of surveying offices, OSM house numbers have a medium positional error of only 11m, whereas house numbers in Google Maps have a medium positional error of 32m.[70]
As already described new mapped objects are available immediately in OSM data, whereas Google Maps is updated daily and official maps several years.[71]

As described in this chapter OpenStreetMap data is not complete in every field, but the level of completeness is increasing, and the quality of OpenStreetMap data is suitable to perform accessibility analyses.

[65] OPENSTREETMAP WIKI: AMBULANCE STATIONS, 2016
[66] HAKLAY, BASIOUKA, ANTONIOU, & ATHER, 2010, p. 321
[67] HAKLAY, BASIOUKA, ANTONIOU, & ATHER, 2010, p. 318
[68] HAKLAY, BASIOUKA, ANTONIOU, & ATHER, 2010, pp. 315-316
[69] LUDWIG, VOSS, & KRAUSE-TRAUDES, 2011, p. 14
[70] AMELUNXEN, 2010, pp. 2-3
[71] WIKIPEDIA:MAP SERVICE COMPARISON, 2017; HECHT, KUNZE, & HAHMANN, Measuring Completeness of Building Footprints in OpenStreetMap over Space and Time, 2013, p. 1072

4 Algorithms

4.1 Routing algorithms

There are several routing services that differ in flexibility and velocity. Therefore it is important to know whether the routing for a project should be fast or flexible to decide which of the existing routing services should be used. The speed or flexibility of a routing service depends on the used routing algorithm. Three important routing algorithms are described in the following chapters.

4.1.1 Dijkstra algorithm

The Dijkstra algorithm is a simple algorithm that computes the shortest path between two nodes in weighted graphs. All kinds of weighted graphs are possible, as long they do not use negative weightings. Distance, time, travel cost or any other measurable values can be set as weight. This algorithm can be used for directed graphs as well as for undirected graphs.

Edsger Wybe Dijkstra developed this algorithm in 1956 to demonstrate the power of a new computer called ARMAC (Automatische Rekenmachine MAthematische Centrum/ Automatic Calculator Mathematical Centre) in a comprehensive form so that all people are able to understand the problem and the answer. Therefore he developed an algorithm that finds the shortest path between two of 64 selected Dutch cities on base of a reduced roadmap of the Netherlands. The algorithm was published in 1959 in the article "A Note on Two Problems in Connexion with Graphs" in the German journal "Numerische Mathematik".[72]

The Dijkstra algorithm groups the nodes in the three types A, B, and C. Type A contains all nodes to which the shortest path from the starting location is known. All nodes that are connected to at least one node of type A are included in type B. Each node of type B can only have a single path (the current shortest path) from the start location to it. The distance of this path is simultaneously the distance of the respective node from the starting location. Finally type C contains all other nodes. At the beginning all nodes are of type C.
Similarly the edges are also grouped in three types (I, II, and III). Type I comprises all edges that are part of the shortest paths from the starting location to the nodes of type A. Edges between nodes of type A and nodes of type B are included in type II (actual shortest paths). All other edges are part of type III. All edges are of type III at the beginning.

First of all the node of the starting location is shifted to type A. A value of 0 is allocated to this node. After that the following two steps are repeated until the destination is reached or the complete graph has been investigated (in case of the destination is not part of the same graph as the starting location).
In the first step all edges that connect nodes of type A with nodes of type B or C are investigated. If a edge connects nodes of type A and B, it is checked whether the newly found path to the node of type B is shorter than the already existing current shortest path. In this case the new path to this node replaces the current shortest path. When an investigated node is part of type C it is shifted to type B and the path to it is computed. Moreover the path added to type II.
After that the node with the shortest distance to the starting location is shifted to type A. Also the

[72] DIJKSTRA, An Interview With Edsger W. Dijkstra, 2001

corresponding edge is moved to type I.[73] Consequently always the shortest path is treated and thereby it is also guaranteed that the shortest path to every node is found.

Because the Dijkstra algorithm is always looking for the shortest path, it searches radially into all directions, beginning from the starting point. It always finds a solution if there is one. Dijkstra is a flexible routing algorithm and therefore not as fast as Contraction Hierarchies.[74] The duration of a route computation increases with the length of the route.[75]

4.1.2 A* algorithm

A* is an improvement of the Dijkstra algorithm and is used in the same way to search the shortest path in a weighted graph. Like Dijkstra it can only use positive weights. In contrast to Dijkstra, A* limits the search by taking the direction to the end point into account. Therefore this algorithm is usually faster than Dijkstra.

The A* algorithm was developed by Peter E. Hart, Nils J. Nilsson and Bertram Raphael in 1968.

A* combines the method of the Dijkstra algorithm (searching for close nodes that can be reached with as less costs as possible from the starting location) with the method of heuristic algorithms like Greed Best-First-Search (searching for nodes that are reachable with as little costs as possible from the destination) to find the node that should be investigated next.
Thereby costs to reach a random node n from the starting location are called g(n), while the "heuristic *estimated cost* from vertex n to the goal" is called h(n).[76] A* searches for the lowest sum of these both values: f(n) = g(n) + h(n). To get a good result a well defined heuristic function is needed.[77] The value of the heuristic function always has to be lower than the total costs of all individual paths. Therefore the beeline is a frequently used heuristic function, because "the shortest road route between any pair of cities cannot be less than the airline distance between them."[78] Because the heuristic function h(n) is freely selectable there are different versions of the A* algorithm. The higher the value for the heuristic function the faster the computation, but also the probability that the shortest path is found decreases. If a value of 0 for the heuristic function is chosen, the A* algorithm is equivalent to the Dijkstra algorithm. When a high value for the heuristic function is chosen, the A* algorithm is equivalent to a heuristic algorithm like Greed Best-First-Search.[79]

A* uses three categories to divide the nodes. These categories are called unknown nodes, open nodes and closed nodes. At the beginning of the algorithm all nodes are unknown. That means no path is found to them so far. Open nodes are all nodes to which a path is known, however it is not certain that it is the shortest path. All open nodes are stored in a list called open list. If the shortest path to a node is known the node is closed and therefore stored in the closed list. Closed nodes are not considered during the computation any more.

The algorithm starts by adding the start location to the open list. Then the f values for all adjacent nodes of the starting node are computed and all selected nodes are added to the open list. Then the node with the lowest f value is selected. If this point is the destination it is added to the closed list

[73] DIJKSTRA, A Note on Two Problems in Connexion with Graphs, 1959, S. 270-271
[74] VELDEN, 2014
[75] RAMM, Routing-Engines für OpenStreetMap, 2017, p. 28
[76] PATEL, 2017
[77] PATEL, 2017; HART, NILSSON, & RAPHAEL, 1968, p. 102
[78] HART, NILSSON, & RAPHAEL, 1968, p. 101
[79] HART, NILSSON, & RAPHAEL, 1968, p. 107

and the algorithm stops. Else the node is added to the closed list and the f values for all adjacent nodes of that point are computed. Again all selected nodes are added to the open list and the node with the lowest f value is chosen. But nodes that are already in the closed list are not investigated any more. Adjacent nodes that are already part of the open list are updated if the new found path is shorter than the old one. This loop is traversed until the destination is reached.[80]

4.1.3 Contraction Hierarchies

Contraction Hierarchies (CH) is the fastest algorithm of thethree algorithms presented in this report to solve a shortest path problem, because by precalculation of the graph it contracts the graph. The precomputation takes a lot of time (depending on the area), but a single query needs just a few milliseconds. However this makes Contraction Hierarchies inflexible, because each time an edge is added, deleted or modified the graph has to be precalculated again. For each routing profile a separate routing graph has to be precalculated because the routing profiles use different weights for the edges.[81] The compiling time for a route increases with route length, but when using Contraction Hierarchies this increase is much smaller than with by A* or Dijkstra.[82]

CH was developed by Robert Geisberger in his diploma thesis "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks" at the University of Karlsruhe (today Karlsruhe Institute of Technology) in 2008.[83]

The algorithm is divided into the two sections contraction and queries. Contraction includes all steps to simplify the graph by contracting it, whereas queries deals with searching for shortest path n based on the simplified graph.
Contraction is done by preprocessing the graph. Depending on the size of the graph this can last hours.
All nodes are evaluated by their "importance" and afterwards ordered regarding this importance. After that from this order a hierarchy is constructed. Less important nodes are "contracted by removing [...] from the network in such a way that shortest paths in the remaining overlay graph are preserved."[84] This means "paths of the form v, u, w [are replaced] by a shortcut edge v, w [...] if v, u, w is the only shortest path from v to w."[84] Figure 18 illustrates this concept.

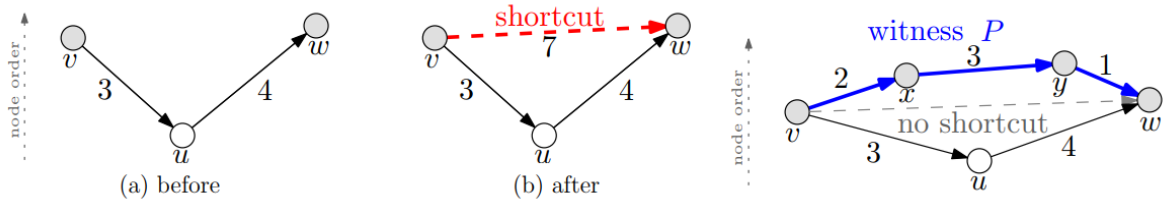

Figure 18: A shortcut node can only be added if there is just one possible way between v, u, w. A witness edge "witnesses that v, u, w is not a shortest path or not the only shortest path."[85]

The computation method for the importance of a node can be freely defined. "However, this choice has a huge influence on preprocessing and query performance."[86] Therefore the computation of the

[80] HART, NILSSON, & RAPHAEL, 1968, p. 102
[81] FUNKE, 2017, p. 5; GEISBERGER, 2008, p. 3
[82] RAMM, Routing-Engines für OpenStreetMap, 2017, p. 2)
[83] GEISBERGER, 2008, p. 1
[84] GEISBERGER, 2008, p. 9
[85] GEISBERGER, 2008, pp. 13-14

importance value in the original CH algorithm is a "linear combination of several priority terms."[86] The definition of these priority terms can be found in the report of Geisberger's diploma thesis.

Shortest path queries are solved by a modified bidirectional Dijkstra algorithm. This is a Dijkstra algorithm that computes the shortest path by simultaneously searching from the starting location as well as from the destination.[87] It also considers the computed importance of the nodes by preferring "edges leading to more important nodes in the forward search and edges coming from more important nodes in the backward search." [88]

An improvement of CH is Customizable Contraction Hierarchies (CCH). CCH reduces the major disadvantage of CH by adding a third section called customization. Contraction is done only for the graph but not for the weights. These are treated in the customization phase. Queries work similar to CH. Therefore after changing the weights, for example for a new profile, the complete graph does not have to be constructed again, just the customization has to be recalculated.[89]

4.2 Other algorithms

In addition to routing algorithms themselves, further algorithms are needed to create the isochrones and to do simplifications if necessary.

4.2.1 CONREC algorithm

CONREC is a subroutine developed in 1987 by Paul Bourke to compute isolines to represent three dimensional information on a two dimensional medium like a map or computer screen. This is also called contouring. The best known use is for computing contour lines, but it can also be used to compute other types of isolines like isobars or isochrones. The third dimension in this case is not the height but the air pressure respective the travel time. In general "some parameter is plotted as a function of two variables, the longitude and latitude."[90] Bourke wrote the subroutine based on a FORTRAN subroutine written by Dr. M. D. Jones of the Auckland University in New Zealand in 1983. The original CONREC subroutine is written in BASIC, but it has already been adapted to many other programming languages like C, C++, C#, Java, and JavaScript. In contrast to other computation-intensive contouring algorithms CONREC "is easy to implement, very reliable, and does not require sophisticated programming techniques or high-level mathematics."[91] In addition to that CONREC has no problems with non-square grids and uneven spacing between coordinates.

CONREC just needs the three-dimensional data and a list of contour levels as input. Typically the contour levels cover the complete range of the investigated parameter for example height, air pressure or travel time (represented on the z-axis).
At the beginning always four adjacent points are grouped to build rectangles. Then each rectangle is split along the diagonals into four triangles. At this stage the z coordinates are ignored. The result of this stage is shown in figure 19a.
Then the coordinates of the center point are computed. The x and y coordinates are interpolated from the four corner points, the z coordinate is the average of the z coordinates of the corner points. As a result there are four three-dimensional triangles. Figure 19b shows one of them. These triangles

[86] GEISBERGER, 2008, p. 14
[87] GEISBERGER, 2008, p. 29
[88] GEISBERGER, 2008, p. 3
[89] DIBBELT, STRASSER, & WAGNER, 2015, p. 1
[90] BOURKE P. , 1987
[91] BOURKE P. D., 1987, p. 143

are later used to compute the isolines, by looking for intersections of each triangle with contour planes that are defined for each isoline level. Along the intersections an isoline segment is drawn as figure 19c shows.
In addition the triangle can intersect the contour plane just in a single point, with its complete area or not at all. Figure 20 lists all six possible intersection types between a triangle and a contour plane. An isoline segment is only drawn in case of a linear intersection (types I to III).
If all intersections between a rectangle and the contour planes have been investigated the algorithm moves forward to the next rectangle until all rectangles have been investigated. The rectangles are investigated from top to bottom and from left to right.
After computation CONREC returns a sequence of line segments together with the corresponding contour levels. This output can be processed by a plotting subroutine to draw the result or other functions that for example combine the individual contour segments into a single contour for each isoline level.[92] As an example of the result of the isoline computation for the complete data and for all isoline levels figure 21 shows the result for one rectangle and one isoline level.

Because CONREC only needs access to the four corner points of a rectangle at the same time it is "suitable for processing large data sets that would exceed the amount of memory available for array storage."[93] It is possible to improve this algorithm, however it gets more complicated and resource intensive.

[92] BOURKE P. D., 1987; BOURKE P. , 1987; NESBITT D. , valhalla/docs/thor, 2017
[93] BOURKE P. D., 1987, p. 144

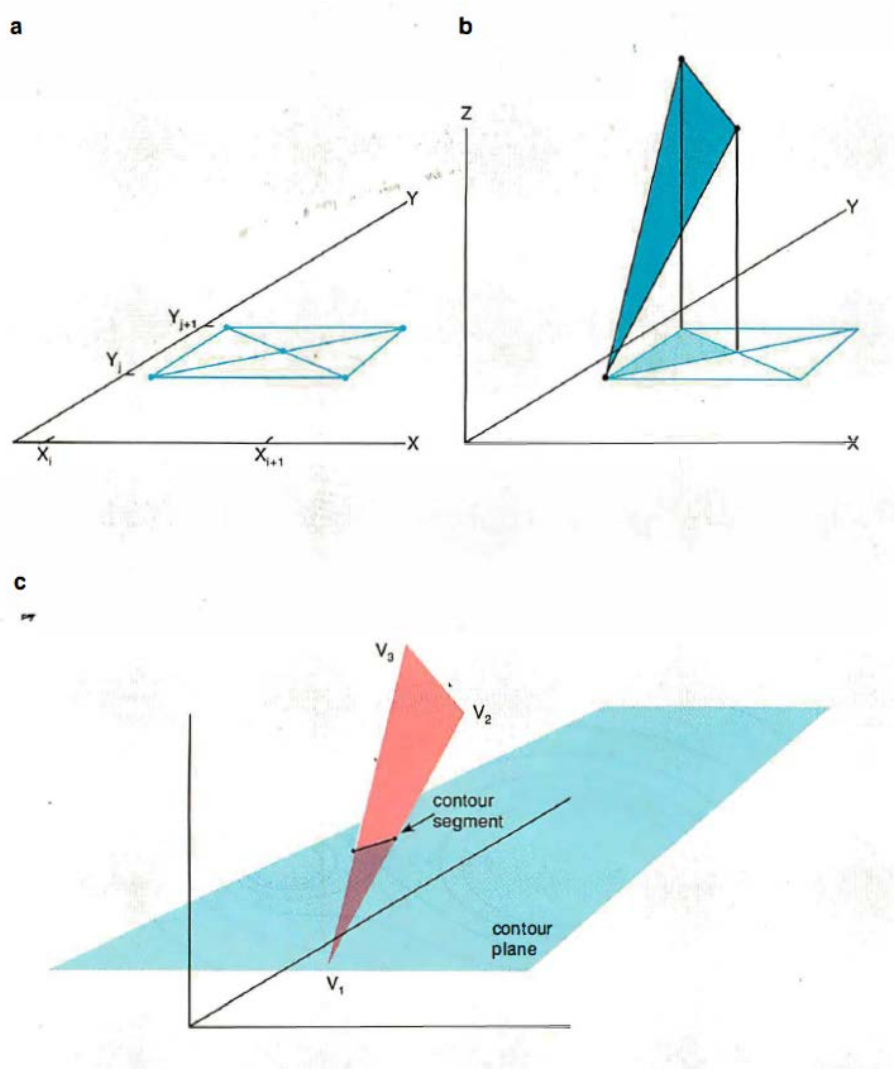

Figure 19: (a) Four adjacent points are grouped to a rectangle (x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j+1}), (x_{i+1}, y_{j+1}). Subsequently the rectangle is further divided into four triangles. (b) After computation of the coordinates of the center points four three-dimensional triangles can be drawn. (c) For each isochrone level there is a contour plane. Isoline segments are drawn if a triangle intersects a contour plane.[93]

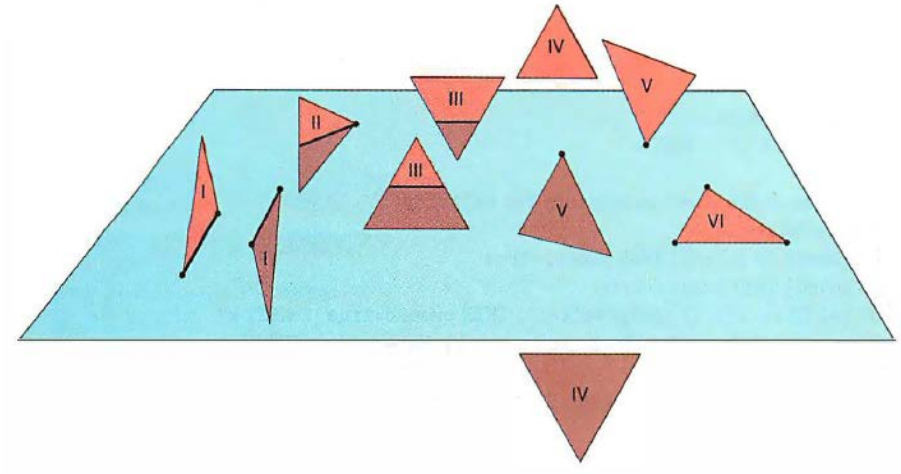

Figure 20: All six possible intersection types of a triangle with a contour plane.[94]

[94] BOURKE P. D., 1987, p. 145

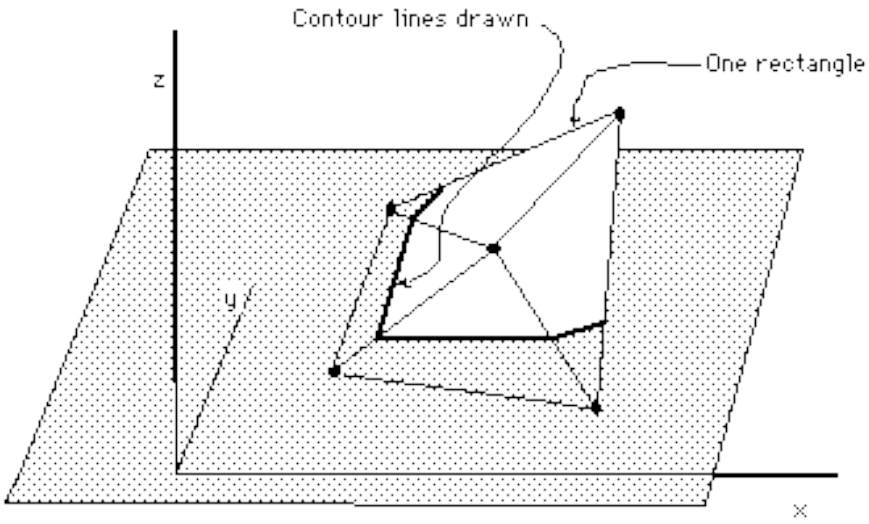

Figure 21: Result of the CONREC algorithm for one rectangle and one contour plane.[95]

4.2.2 Douglas-Peucker algorithm

The Douglas-Peucker or Ramer-Douglas-Peucker algorithm is used for generalization through polylines simplification. This is done by reducing the number of points in the polyline, more precisely by selecting characteristic points.

David Douglas and Thomas Peucker developed this algorithm in 1973 based on another algorithm developed by the German company AEG that was used in their Experimental Cartographic Unit for printing. Douglas-Peucker was originally coded in FORTRAN IV and ALGOL W.[96]

The AEG algorithm deletes points if they are closer to a straight line between two points than a predefined tolerance. Therefore the first point of a polyline is defined as representative point and connected with "straight lines to subsequent points until one point between the representative point and the sub-point is further away from the line linking the two than a pre-set tolerance value." [97] If an investigated point is further away from the straight line than the tolerance, "the point before the sub-point becomes a new representative point."[97] This procedure is repeated until the end of the polyline is reached.[97]

In contrast to the AEG algorithm Douglas-Peucker uses the first and the last point of a polyline to start the calculation and select points.
The first point is called anchor and the last point is called floating point. Anchor and floating point are connected by a straight line. All points in between are investigated to find the point with the largest perpendicular distance to the straight line. If the distance between this point and the straight line is larger than a predefined tolerance the point is set as new floating point, if not the polyline can be generalized by the straight line. All intermediate floating points are stored. In case the floating point has moved this procedure is repeated, otherwise the floating point becomes the new anchor and the last intermediate floating point is set as new current floating point. The generalized polyline consists of all points that were defined as an anchor.[98] Figure 22 illustrates the process of the algorithm with

[95] BOURKE P. , 1987
[96] DOUGLAS & PEUCKER, 1973, pp. 116-117
[97] DOUGLAS & PEUCKER, 1973, p. 116
[98] DOUGLAS & PEUCKER, 1973, p. 117

its different steps. In addition to that figure 23 shows the original polyline at the top and below the generalized lines with the respective tolerance distance in front of them.

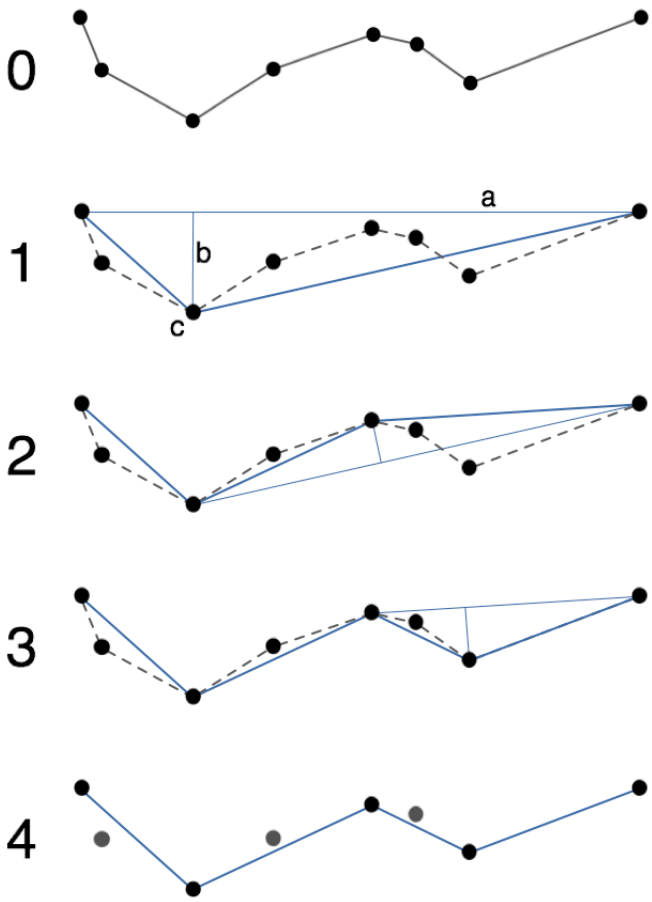

Figure 22: Process of the Douglas-Peucker algorithm. Polyline 0 is the original. In the first step (1) both endpoints of the polyline are connected by a straight line "a" and the point with the largest perpendicular distance "b" to this line is identified (point c). In the next step (2) point "c" is the new endpoint if the measured distance is larger than the predefined tolerance. Also the procedure is repeated. Again a new point that lies further away from the straight line as the tolerance value is identified (3). Finally the lowermost picture shows the final result.[99]

The name of the algorithm is inspired by the names of two authors of the article "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature" in The Canadian Cartographer in 1973. However, because Urs Ramer had published the algorithm a year before them in Computer Graphics and Image Processing, therefore the algorithm sometimes is also called Ramer-Douglas-Peucker.[100]

[99] WIKIPEDIA: Douglas-Peucke, 2016
[100] RAMER, 1972

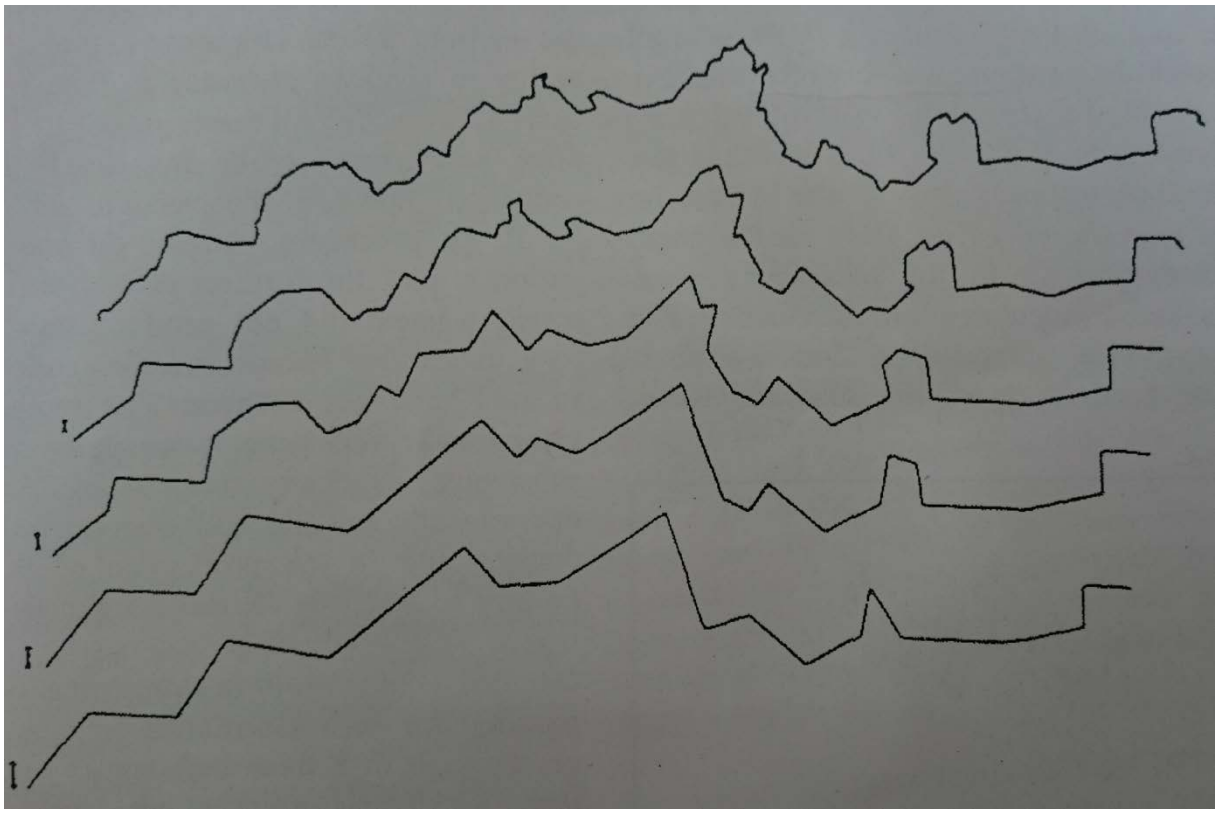

Figure 23: Results for the generalization of the polyline on the top with different tolerance values by the use of the Douglas-Peucker algorithm. The corresponding tolerance value for each result can be seen left of the individual result.[101]

[101] DOUGLAS & PEUCKER, 1973, p. 119

5 State of the art methods of accessibility analysis

There are different methods to define the reachable area around a fire or a rescue station. The reachable area is the area which contains all the points which can be reached within the travel time period. Different methods and examples for their implementation are explained in this chapter.

5.1 Circle-/beeline-method

A simple but imprecise approach is the use of circles around the stations. It can be called "circle" or "beeline method", because the radius of these circles defines the distance an emergency vehicle is able to cover in a certain period. Figure 24 shows an example of the volunteer fire brigade of Finnentrop.

To compute the distance for the radius an average velocity for the emergency vehicle and the duration of the travel time period have to be determined first. The average velocity depends on the vehicle: car, van or truck. Standard values for trucks with special signs (blue flashing lights and sirens) are 40 km/h in town and 60 km/h out of town.[102] In addition to that the travel time depends on the location, because in each country or even state or province the time target is different. For example in Saxony the target is 4 minutes for volunteer fire brigades and 8 minutes for career fire brigades. They differ because the turnout time of professional fire brigades is 1 minute (they are already at the fire station) and for volunteer fire brigades it is 5 minutes (they have to drive to the fire station first).[103] By using the formula s = v * t (distance = velocity * time) the radius can be computed. Finally the location of the stations and the circles around them are drawn into a large-scale map (about 1:50,000).

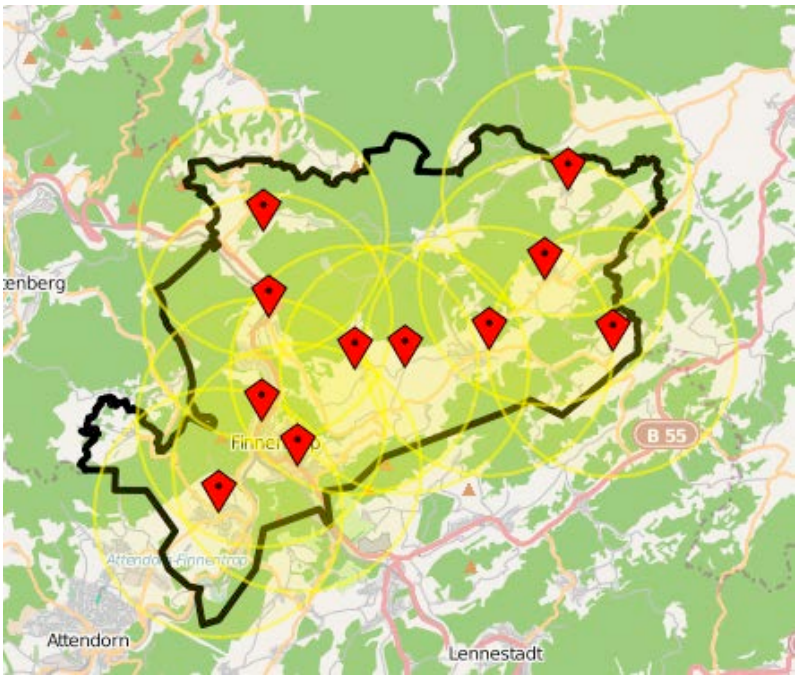

Figure 24: Definition of the reachable region with the circle or beeline method of Finnentrop in North Rhine-Westphalia (Germany).[104]

[102] SÄCHSISCHESES STAATSMINISTERIUM DES INNERN, 2005, p. 1171; LANDESFEUERWEHRVERBAND HESSEN, 2015, p. 8
[103] SÄCHSISCHESES STAATSMINISTERIUM DES INNERN, 2005, p. 1171
[104] NEIS, 2010; FREIWILLIGE FEUERWEHR FINNENTROP, 2010

But this method ignores the street network and the different road categories. Therefore this method can lead to unrealistic results. Non-reachable areas (e.g. a river without bridges is between an area and the station or a mountain region which is only reachable by winding roads) will be marked as reachable within a certain number of minutes. Moreover other regions, which are reachable in time by using for example a motorway, are marked as not reachable within the period.
An improvement of this method is to cut the circle along insurmountable barriers like rivers, railways, motorways.[105]

5.2 Road Axis-Method

The Road Axis-Method leads to polygonal instead of circular areas. It is more resource-intensive to compute, but yields more precise results. For this method the travel time is calculated along the roads. This method can be implemented in three different ways.

5.2.1 Manual methods

5.2.1.1 Opisometer method

The first way is an advancement of the beeline-method. With the use of an opisometer (see figure 25) the computed distance is measured along the separate road axes beginning at the fire or rescue station. The endpoints of all possible routes are marked on a map and are later on connected. The result is a polygon which represents the reachable area. However this approach still ignores different road categories.[103]

Figure 25: Mechanical opisometer in use.[106]

5.2.1.2 Test drive method

The second way to get the reachable area are test drives. That means time and distance are measured either during real emergency drives or during test drives with an ordinary car. For example

[105] LANDESFEUERWEHRVERBAND HESSEN, 2015, p. 8
[106] WIKIPEDIA: OPISOMETER, 2016

the reachable area which can be reached by using an ordinary car is comparable to the range which is reachable with a truck with special signs. Just like in the first approach the endpoints are marked and connected to get the polygon of the reachable area.[103] On the one hand, this method delivers the travel time under real conditions. Moreover it considers the velocity changes because of the course, the width or the category of the road. Of course test drives are only a snapshot, because they depend on weather and traffic conditions. To get verifiable values several measurements are needed to compute the average. On the other hand these kind of measurements are time, cost, and work intensive, because measurements of emergency drives are not available for all routes. Each possible route has to be measured in real time, which takes 4 to 8 minutes per route plus the way back to station, also fuel costs, and other operating costs for the test vehicle will occur. This method works with just a few measurements, but the more measurements are done, the more precise the result will be. For example the career fire brigade of Wiesbaden in the German federal state of Hesse has used this method to generate the isochrones around their fire stations.[107]

5.2.2 Digital non proprietary methods

In contrast to the approaches mentioned so far this way uses a computer to estimate the reachable area. With this approach it is also possible to take care of different road categories and the course of the roads as long as the available data of the road network contains this information and the software or service used is able to deal with it. In comparison to the test drives this method causes less costs and is much faster. Figure 26 shows an example from the volunteer fire brigade of Finnentrop for the use of OpenRouteService (explained in chapter 5.2.2.2).
There are a lot of different software and routing services solutions to do so. For both kinds there are free or proprietary versions or solutions available (they are described in chapter 2.5.2.3). Moreover OpenStreetMap offers a worldwide road network on which the accessibility analysis can be performed.

Software like GIS systems (proprietary ones like ESRI ArcGIS as well as open source ones like QGIS) have the disadvantage that they have to be installed on the user's computer first, before the user is able to do an analysis for a limited area. The base map for this area has to be downloaded first and also stored on the user's computer and the user has to know how to create an accessibility analysis in this software. Sometimes it is necessary to install extensions of the software before it is feasible to do an accessibility analysis.

Routing services can be used both installed on the user's computer and as a service in the internet. While installing a routing service on the user's computer has similar disadvantages as a software, the use of a service on the internet is easy and uncomplicated for the user. This is because the user just has to select a few settings, the analysis then will be done automatically and finally the result will be shown in a map. These settings are to choose a starting point or the starting points, the duration or distance (sometimes also definition of a number of time step or a step interval is possible) and the profile (e.g. bicycle, car, truck). This method also allows a much larger coverage up to worldwide coverage.
Although there are a lot of different routing services, not all of them support isochrones or are open source.
Several examples of routing services using OpenStreetMap as the base map and offering isochrones are listed in the OpenStreetMap Wiki (https://wiki.openstreetmap.org/wiki/Isochrone and

[107] PITTHAN, 2017

https://wiki.openstreetmap.org/wiki/Routing), but most of them are not open source. There are also a few routing services which are partially open source. They offer a base open source version with restricted functionalities and building on this API-keys are needed to use extra functions. Some of these programs are also developed for a special task and therefore support only a few profiles, use just the issue-specific part of the OpenStreetMap data or use additional data.[108] Examples for routing services which offer isochrones are GraphHopper, Open Source Routing Machine (OSRM), and Valhalla just to name a few of them. In the following sub chapters a couple of routing services are examined.

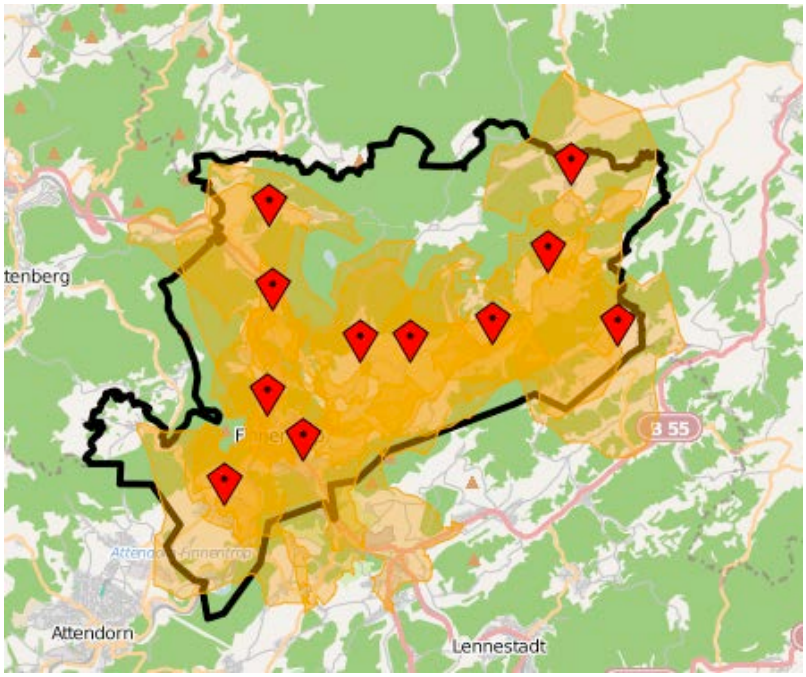

Figure 26: Definition of the reachable region with the polygon or road axis method of Finnentrop in North Rhine-Westphalia (Germany). The accessibility analysis was done by using OpenRouteService.[104]

5.2.2.1 GraphHopper Isochrone API

GraphHopper is an open source routing engine that can be used as library, as an online service called GraphHopper Maps (https://graphhopper.com/maps) or as routing application programming interface API over HTTP. Besides ordinary routing it offers elevation information, isochrone computation, geocoding, and navigation. For each of them an extra API is provided. For using the API credentials are required, these can be requested at https://graphhopper.com/dashboard/#/register.

GraphHopper is able to compute routes very fast and efficiently, is very robust and easily customizable. It can be used for routing (also indoor), solving vehicle routing problems, vehicle tracking, traffic simulation, isochrone calculation, and virtual reality games.[109] According to Peter Karich, the author of GraphHopper, GraphHopper "is comparable fast to Google or Bing Maps, but if you use it in a LAN or directly from within your application it is a lot faster."[110]

[108] OPENSTREETMAP WIKI: ROUTING, 2016; OPENSTREETMAP WIKI: ISOCHRONE, 2017
[109] KARICH, GraphHopper an Open Source and Flexible Alternative to Google Maps API, 2013a; KARICH & LEGRAIN, GraphHopper Routing Engine, 2017
[110] KARICH, GraphHopper an Open Source and Flexible Alternative to Google Maps API, 2013a

It was primarily developed by Peter Karich and is still under active development. The first version was released on 22 July 2013 after more than one year of development.[111] Since 2016 GraphHopper is maintained by the GraphHopper GmbH.[112] The recent version 0.9.0 was released on 31 May 2017.[113]

The source code of GraphHopper is written in Java and available on GitHub (https://github.com/graphhopper/graphhopper). GraphHopper is designed to be usable on servers, desktop (Linux, Mac, Windows), and mobile devices (Android, iOS).
The GraphHopper API is customizable. Thus, for example the OpenStreetMap data by default used for routing and Shuttle Radar Topography Mission (SRTM) data for the elevation can be replaced by other data sources. GraphHopper provides routing options for bicycle, car, and pedestrian by default, but it is possible to add new ones.[114] The user also can choose between the shortest and the fastest path. The computation of them can be influenced by adapting the weightings. With this it is also possible to add weights for other topics like travel cost, elevation or suitability.

Because of the use of Contraction Hierarchies for computation GraphHopper offers extremely fast route computation (called speed mode). If necessary the routing algorithm can also be changed to Dijkstra or A* (called flexible mode). Starting with software version 0.9.0 there is also a third mode called hybrid mode that tries to combine the advantages of both.[115]

GraphHopper uses the Apache License 2.0. Therefore everyone can use and adapt GraphHopper in their own projects, it does not matter whether they are free or commercial, open source or closed source. The complete terms of license can be found at the Apache website (https://www.apache.org/licenses/LICENSE-2.0).[114]

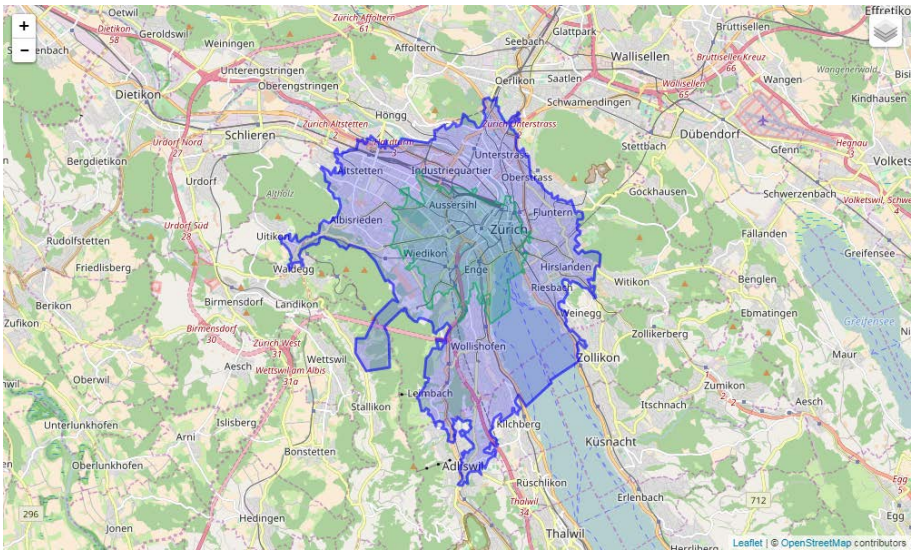

Figure 27: Example result of GraphHopper Isochrone API. The inner isochrone is a 5 minutes isochrone and the outer isochrone is a 10 minutes isochrone. Start point is the station South of the career fire brigade Zurich, Switzerland. [116]

To calculate isochrones the GraphHopper Isochrone API is needed. Besides defining the starting point it is possible to choose a time or distance limit and a vehicle (routing option) for the isochrone

[111] KARICH, GraphHopper Maps 0.1 – High Performance and Customizable Routing in Java, 2013
[112] GRAPHHOPPER, A
[113] KARICH, GraphHopper Routing Engine 0.9 Released, 2017
[114] KARICH & LEGRAIN, GraphHopper Routing Engine, 2017; OPENSTREETMAP WIKI: GRAPHHOPPER, 2017
[115] KARICH, Flexible routing at least 15 times faster, 2017
[116] GRAPHHOPPER, B

computation. In addition to that the time or distance steps (called buckets) and the travel direction (which area is reachable from the starting point in a certain time or distance or from which area the "starting" point is reachable in a certain time or distance) can be defined. The response is in GeoJSON format.[117] An example of an isochrone computation for the station South of the career fire brigade Zurich made with GraphHopper Directions API Live Examples (https://graphhopper.com/api/1/examples/#isochrone) can be seen in figure 27. At the moment the source code for computing isochrones is not open source but it is planned to be made public, maybe in the next release.[118] The maximum time for which isochrones can be computed depends on the chosen service package between 20 and 60 minutes, the free package does not include the Isochrone API.[119] The isochrone interval is freely selectable.[117]

5.2.2.2 openrouteservice.org

The online routing engine OpenRouteService uses open source software, open standards published by the Open Geospatial Consortium (OGC) and open data (OpenStreetMap) for its calculations. Only the frontend is open source and can be found on GitHub (https://github.com/GIScience/openrouteservice). It is available under the MIT license, therefore everybody is allowed to use this without charge and any restriction (for further detail see the license at https://opensource.org/licenses/MIT).[120] Because of limited resources an API key is needed to send requests to the hosted backend.[121] Registration for this API is possible at https://developers.openrouteservice.org/portal/register. There is only a free API version with a limited number of requests. All features are useable via the official OpenRouteService website www.openrouteservice.org.[122]

OpenRouteService was developed and is still pursued by several members of the University of Heidelberg, Germany (Pascal Neis, Alexander Zipf, Maxim Rylov , Lu Liu, Timothy Ellersiek, Hendrik Leuschner, Amandus Butzer, and Adam Rousell). The first version was released in April 2008.[123] On 29 April 2017 the latest version, 4.1, was released.[124]

It is possible to do routing and geocoding as well as computing isochrones and searching for POIs nearby. OpenRouteService offers the four services Directions, Geocoding, Isochrones, and Locations. Areas that should be avoided can be defined by polygons. By default OpenRouteService offers a lot of different profiles for bikes, cars, pedestrian, trucks, and even wheelchair.[123] These profiles can be adapted to the personal needs by parameterizing the API request.[125]

The backend offers OpenStreetMap data worldwide and is updated weekly.[126] For calculations OpenRouteService uses the A* algorithm. Developers can interact with the backend through HTTP GET requests.[121] The response of the backend will be in GeoJSON format.[124]

---

[117] KARICH, Isochrone API, 2017
[118] KARICH, Isochrone in open source version of graphhopper?, 2017
[119] GRAPHHOPPER, C
[120] MCCAULEY & GRI, 2017
[121] OPENSTREETMAP WIKI: OPENROUTESERVICE TALK, 2014
[122] OPENROUTESERVICE: APIS
[123] OPENSTREETMAP WIKI: OPENROUTESERVICE, 2017
[124] ELLERSIEK, 2017; SWAGGERHUB, 2017; OPENROUTESERVICE: API FEATURES
[125] BUTZER, 2017
[126] RYLOV, 2015

OpenRouteService is able to execute an accessibility analysis in respect to travel time or distance for up to five start locations at once. This kind of computation is possible for all offered profiles. Isochrone computation is limited to a maximum number of 10 intervals, a maximum distance of 120 kilometers, and a maximum time of 60 minutes. For each start location OpenRouteService returns the isochrones as contour line of filled polygons. If two or more of them are intersecting each other they are not merged, but displayed on top of each other.[127] Figure 28 shows the result of the computation of the five and 10 minutes isochrone for the station South of the career fire brigade Zurich again, this time computed with OpenRouteService.

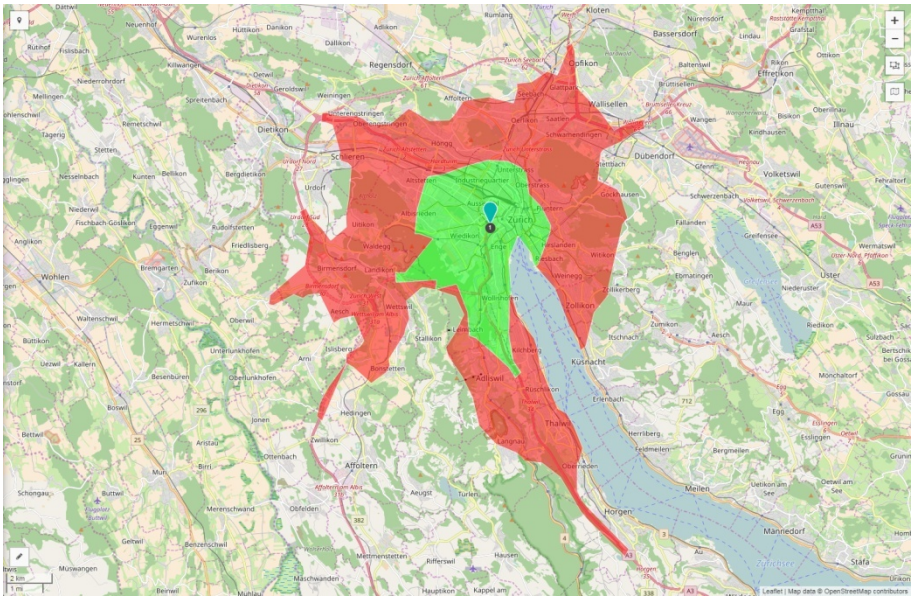

Figure 28: Example result of OpenRouteService. The inner polygon is a 5 minutes polygon and the outer polygon is a 10 minutes polygon. Again the start point is the station South of the career fire brigade Zurich, Switzerland. [128]

5.2.2.3 OSRM-isochrone

OSRM-isochrone is an OSRM add-on and computes isochrones via the Turf.js-isolines library on the base of OSRM files (*.osrm). It is open source and uses open source libraries, software, and data.

The first version 1.0.0 of OSRM-isochrone was released on 1 December 2014. On 26 October 2016 the latest version 3.0.0 was released.[129] OSRM-isochrone is maintained by Mapbox, an American business working with OSM data. Like OpenRouteService OSRM-isochrone is published under the MIT license. The source code can be found on GitHub (only version 2.0.0) as well as npm (https://github.com/mapbox/osrm-isochrone or https://www.npmjs.com/package/osrm-isochrone).

Before computing isochrones the OSRM files have to be preprocessed. To compute OSRM files the Open Source Routing Machine (OSRM) has to be installed. Alternatively the enclosed binary files of OSRM can be used. OSRM files are based on OpenStreetMap data and use Contraction Hierarchies. To compute the isochrones beside the start location the time for the isochrones, the resolution of the grid (generalization), and a maximum speed can be determined. The response will be in GeoJSON format. [130] An example result of this isochrone computation algorithm can be seen in figure 29.

[127] SWAGGERHUB, 2017; OPENROUTESERVICE: APIS; OPENROUTESERVICE: API FEATURES
[128] OPENROUTESERVICE: ISOCHRONES
[129] NESBITT A. , 2017
[130] HERLOCKER, GIRARD, NIKLAUS, VAN DOORN, & RODRIGO, 2016

OSRM-isochrone is no longer maintained, all dependencies are out of date.[131] Also the GitHub repository of OSRM-isochrone indicates no new activities since around the release of the latest version in October 2016.

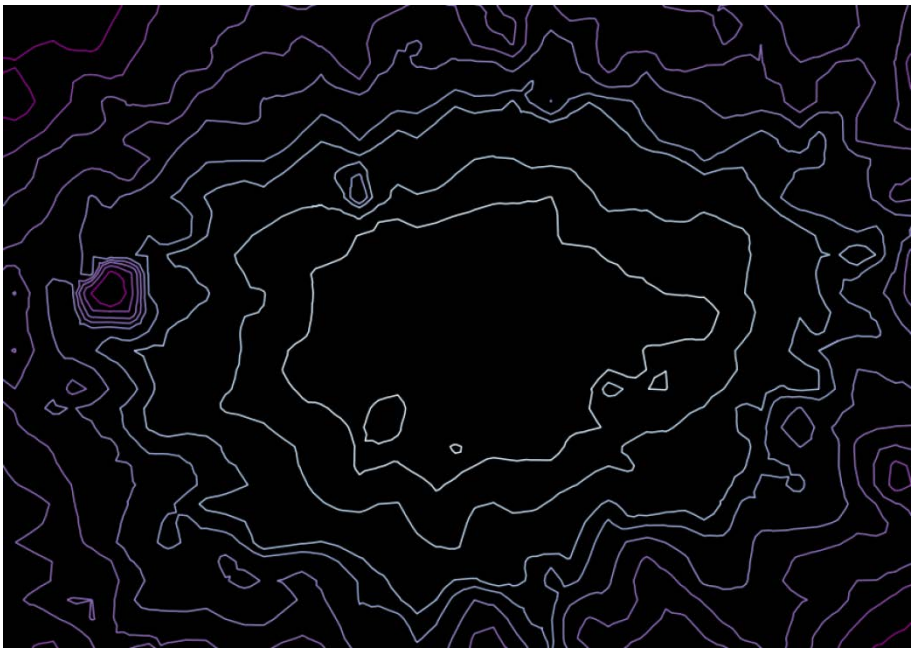

Figure 29: Example of the OSRM-isochrone GitHub repository.[130]

5.2.2.4 Valhalla

Valhalla is a free and open source routing engine that uses open source data, especially from OpenStreetMap. It can be self hosted or used via an API that is called Mapzen Turn-by-Turn. No matter which solution is chosen for computing, the response can be embedded through Leaflet into a website. The C++ source code and a description of Valhalla can be found on GitHub (https://github.com/valhalla).[132]

Routing with trip information in many different languages, computation of isochrones, solving the travelling salesman problem, map matching, and navigation is possible worldwide with Valhalla. Valhalla offers several routing profiles like bicycle, car or pedestrian. As a special feature a multimodal profile is offered that combines public transport (buses, ferries, trains, and subways) with walking. Information about public transport like routes, stops, and timetables is obtained from Transitland, an open transit data directory (https://transit.land).[133] Custom profiles can be added.[134] There is an online example of public transit routing, isochrone computation, and map matching at https://mapzen.com/mobility/explorer.

Route computation is done by an adapted A* algorithm by using JSON respective GeoJSON for input and output.[135] In contrast to that the Dijkstra algorithm is used for isochrone computation.[136] For elevation ETOPO1, GMTED, NED, and SRTM data is used.[137]

[131] VERSION EYE GMBH
[132] FOSTER, et al., 2017
[133] MAPZEN: MULTIMODAL TRANSIT ROUTING, 2017
[134] NESBITT D. , Valhalla Open Source Routing, 2015
[135] NESBITT & WILTON, THOR - DETERMINING THE BEST PATH, 2017
[136] NESBITT D. , What are Isochrone Maps?, 2017

Valhalla was announced on 20 March 2015, launched on 04 June 2015 and is still under active development.[138] The latest version 2.3.5 was released on 22 August 2017.[139] Valhalla is developed among others by a core team of Mapzen, an American mapping subsidiary of Samsung. The source code of the Valhalla project is available under the MIT license, which allows everybody to contribute and improve Valhalla.[132]

Valhalla consists of several parts that are responsible for different tasks. A list of modules and their tasks can be found at https://github.com/valhalla/valhalla/tree/master/docs. The project name as well as the names of its parts originate from the Norse mythology.

Isochrone calculation can be influenced by selecting a start location, a profile, a list of minutes for which isochrones should be computed and optically the color in which they should be displayed and a point of time in case of multimodal profile. In addition to that the result can be displayed either as isochrones or as filled polygons, small fragments can be filtered out, and the isochrones can be generalized. If the Mapzen Turn-by-Turn API is used also an API key is mandatory. Then also the maximum number of isochrones is limited to four. The response is in GeoJSON format and can be displayed on a map.[140] An example of isochrone computation with Valhalla can be seen in figure 30.

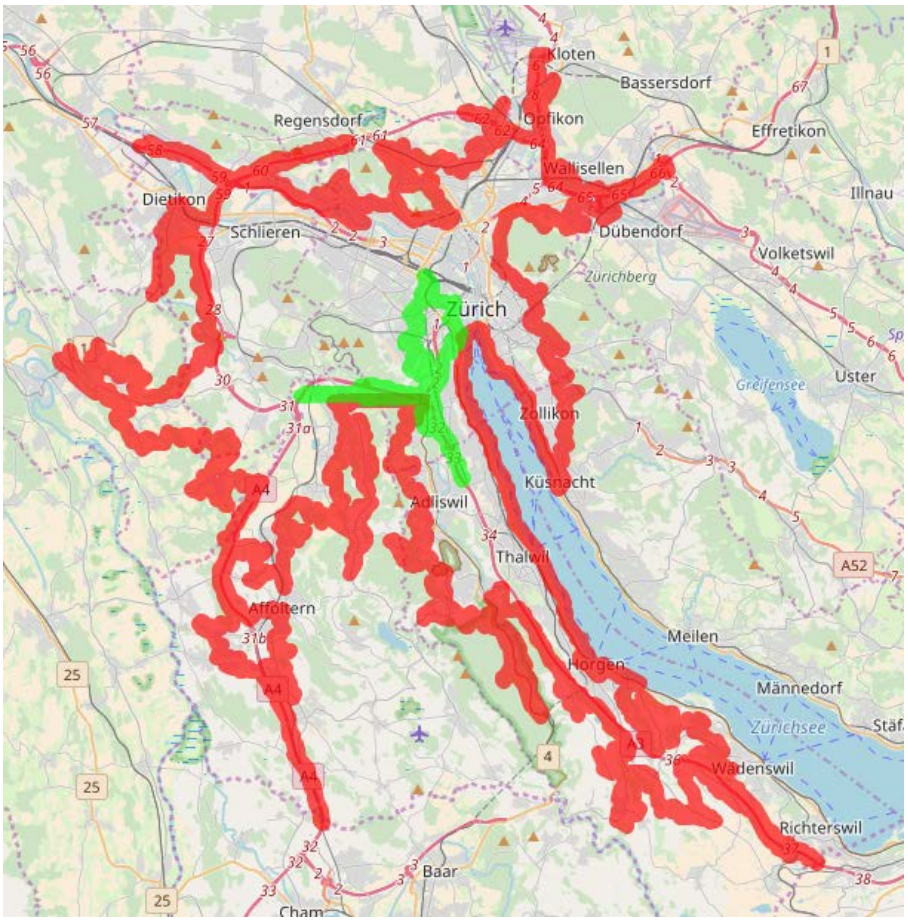

Figure 30: Example result of Valhalla. The inner isochrone is again a 5 minutes isochrone and the outer isochrone is a 10 minutes isochrone. Start point is the station South of the career fire brigade Zurich, Switzerland.

[137] MAPZEN: ELEVATION, 2017
[138] NESBITT D. , Valhalla Open Source Routing, 2015; DILUCA, GEARHART, KNISELY, KREISER, & NESBITT, Introducing Valhalla, 2015
[139] DILUCA, GEARHART, KNISELY, KREISER, NESBITT, & WILTON, Release notes, 2017
[140] MAPZEN: ISOCHRONE, 2017

5.2.3 Digital proprietary methods

Digital proprietary methods are mentioned just for the sake of completeness because they do not conform to the predefined aims in chapter 1.2. As one example of a digital proprietary method the computation of isochrones via ESRI ArcGIS is described.

Computing Isochrones is possible by using ArcGIS Online as well as by using ArcGIS Desktop. However ArcGIS Desktop is more powerful, therefore only this way is described. It is possible to do this calculation for different profiles like car, pedestrian or truck (called travel modes). These profiles can also be adapted or new profiles can be added. Calculations are possible for maximum five hours and maximum 300 miles (around 480 kilometers).

To compute isochrones using ESRI ArcGIS Desktop the tool Service Area of the ArcGIS extension Network Analyst is necessary. After creating a "New Service Area" it is possible to import the locations of the stations (up to 1,000) that represents the start locations for the calculation later on. The definition of the maximum travel time as well as time levels in between is adjustable by default Breaks in the Analysis Settings. Finally the Solve button has to be clicked to start the computation. Then the result can be solved by the Export Data function. It is also possible to compute the isochrones depending on the day and time of day if the available data offers this information. A detailed description of the workflow can be found at http://desktop.arcgis.com/en/analytics/case-studies/which-areas-are-within-four-minutes-of-a-fire-station.htm?lg=en.

For computation of the travel time ArcGIS uses the Dijkstra algorithm and stores the travel time in a Triangulated Irregular Network (TIN). The travel time corresponds to the height of the TIN at a certain location.[141]

[141] ESRI, 2017; ARCGIS ONLINE HELP, 2017

6 Concept

6.1 Target definition

The calculation of the accessible area originating from a starting point (emergency service station) is easily achievable for the user through the use of a web interface. The user only has to choose between a few options and to define the start point by clicking into the map at the desired location. A suitable routing backend deals with the computation of the isochrones on the fly and the presentation of the result on the web interface (frontend) again. Examples for the aforementioned settings are the choice of a vehicle profile, a maximum travel time, and the definition of a name for the computation. The user has the possibility to store, load, and merge results. While saving a result to a GeoJSON file the defined name is used to save the file. Although it is technically feasible to do a accessibility analysis computation with several starting points it is not provided in this work to allow the user to set different settings for each starting point. To make it still possible to generate a single GeoJSON file for a city with more than one station the user is able to combine several GeoJSON files into a single one with a merge functionality. In addition to that it is possible to check whether it is possible to observe the response time in a defined area or not by using a inspection function. This function also allows the user to identify undersupplied regions. The inspection can be done through simultaneous display of the isochrones and the borders of the operation area. Finally a clear function is available to reset all settings to their default values. It is also possible to add further settings if the chosen backend supports them.

6.2 Graphical User Interface (GUI)

The graphical user interface as frontend allows the user to interact with the routing service for isochrone computation as backend. It enables the user to change the settings to influence the isochrone computation without knowing anything about the operating principles of the backend. In internet technology the GUI is synonymous to the website.

Zooming and panning functions are standard for web maps. For an easy and comfortable use the design and the structure of the website also have to be plain, simple, and clearly defined. The handling has to be intuitive and possible without many words to allow fast using without confusions. It is also an advantage if a website can be used with just a few interactions to get a result. It is also very important to make the map section as large as possible to ensure good usability. All modern online map services like OpenStreetMap or Google Maps use the full size of the browser window. In addition to that it is also necessary to display the scale of the map to enable the user to gauge distances. Last but not least it is handy to have a function to clear the map and reset the website to the default values.

The website is designed according to these specifications. The map section covers the complete browser window and the controllers are placed at the edges of the window. Only the most important controller for zoom and search a location as well as a scale bar and the copyright are added directly on top of the map. All others are combined into a menu. Figure 31 shows a sketch for the GUI. Selecting the menu controller expands a sidebar with several tabs to change the isochrone computation method, to handle file, and checking the result as well as display the imprint. Sketches for the tabs are shown in figure 32 to 34. The first tab is called "Isochrones" and contains all controller to adapt the computation of the isochrones. Controllers to clear the map, for file handling and for checking the result are contained in the second tab, which is called "File". By default the first tab is active.

As explained before each tab includes several controllers to allow the computation of the isochrones, handling files or checking the result. Depending on the type of setting different controllers are used. To enable the user to choose between a couple of predefined options like the vehicle profiles a drop-down menu is used. In this menu all available profiles are listed and if the user selects one of them, it becomes active. Input fields are used to allow the user to define a name for the computation, search for a city to check the accessibility and define the travel time. With the help of buttons the user is able to call functions to clear the map or to handle files.

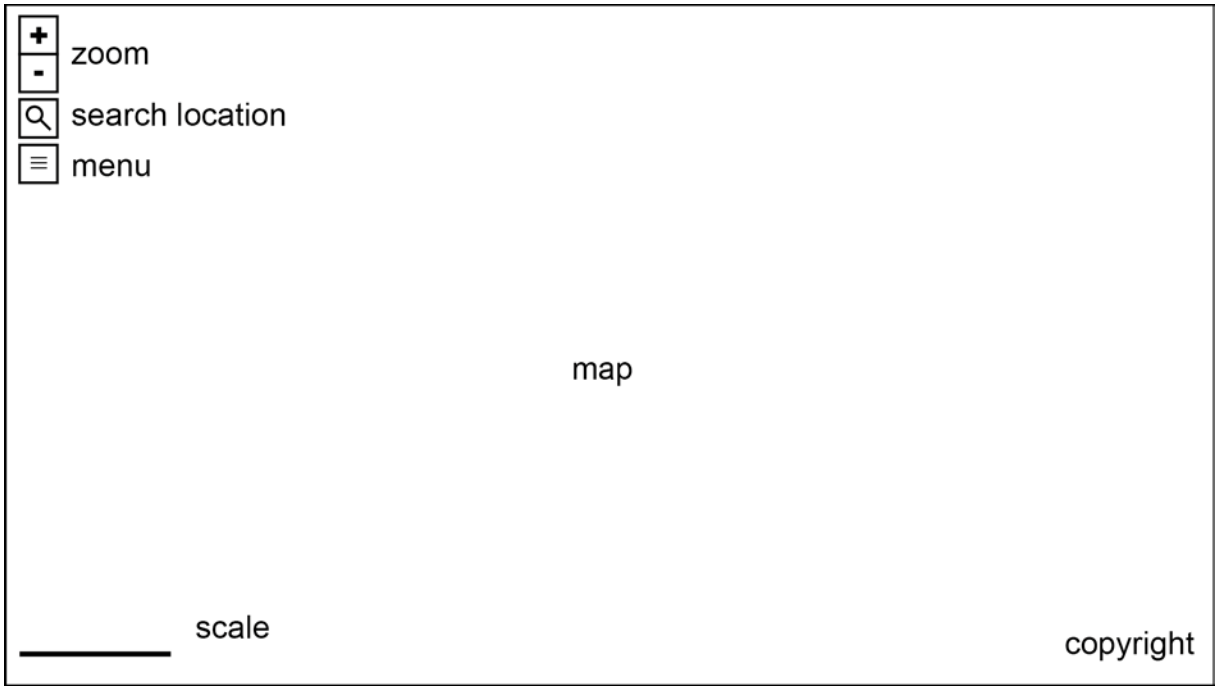

Figure 31: GUI of the website. The complete browser window is covered by the map. Some controller are added at the upper left corner. A scale bar and the copyright are presented at the bottom corners. A click at the menu buttons opens a menu with further settings options.

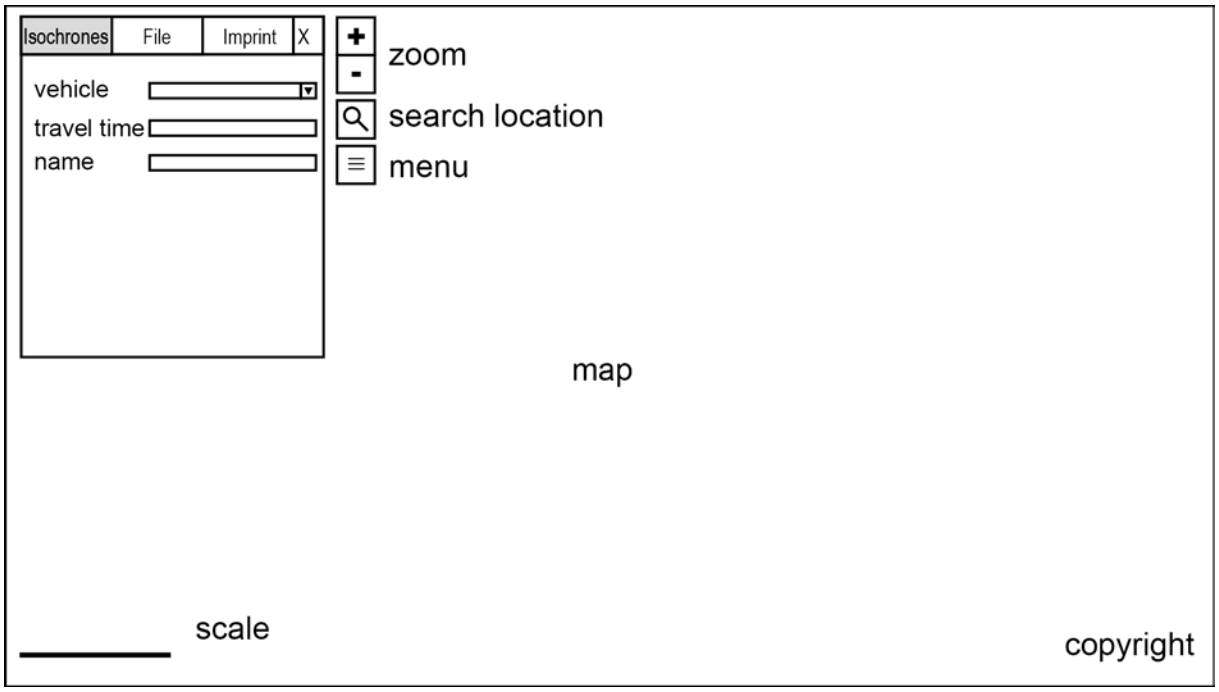

Figure 32: Extended sidebar. By default the Isochrones tab is displayed where the user can choose a routing profile, the travel time, and a name.

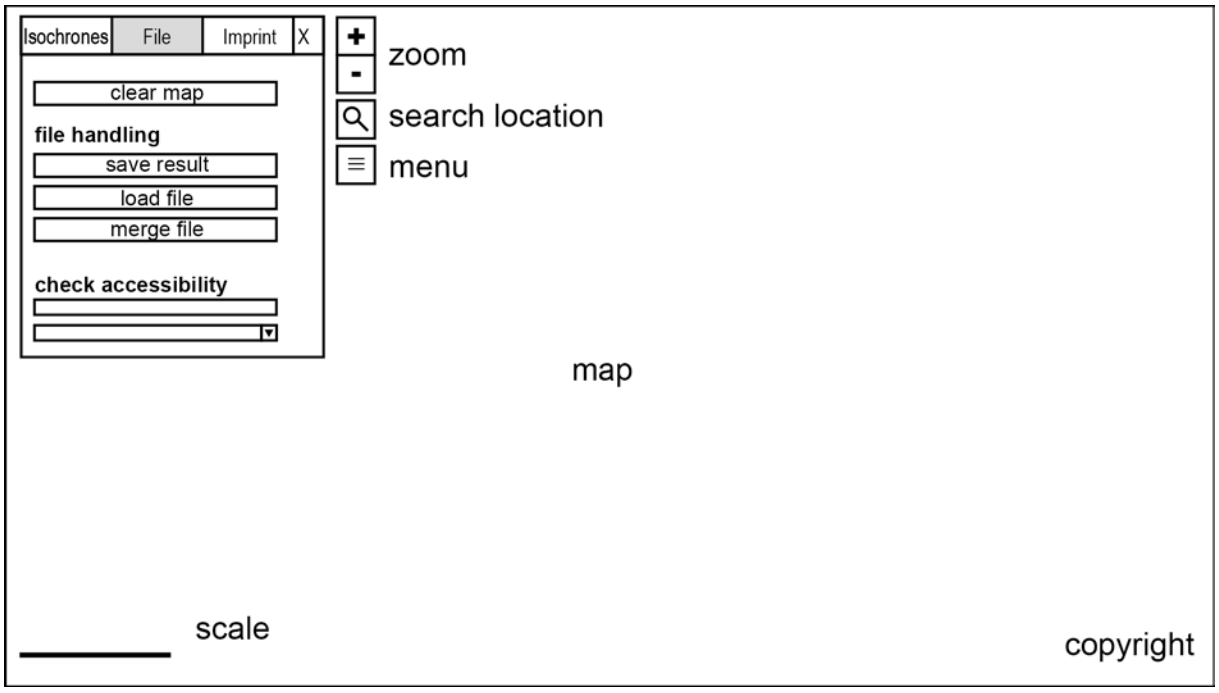

Figure 33: In the File tab the user is able to reset the map to the default values, handle files, and check if the complete area is accessible in a defined time interval.

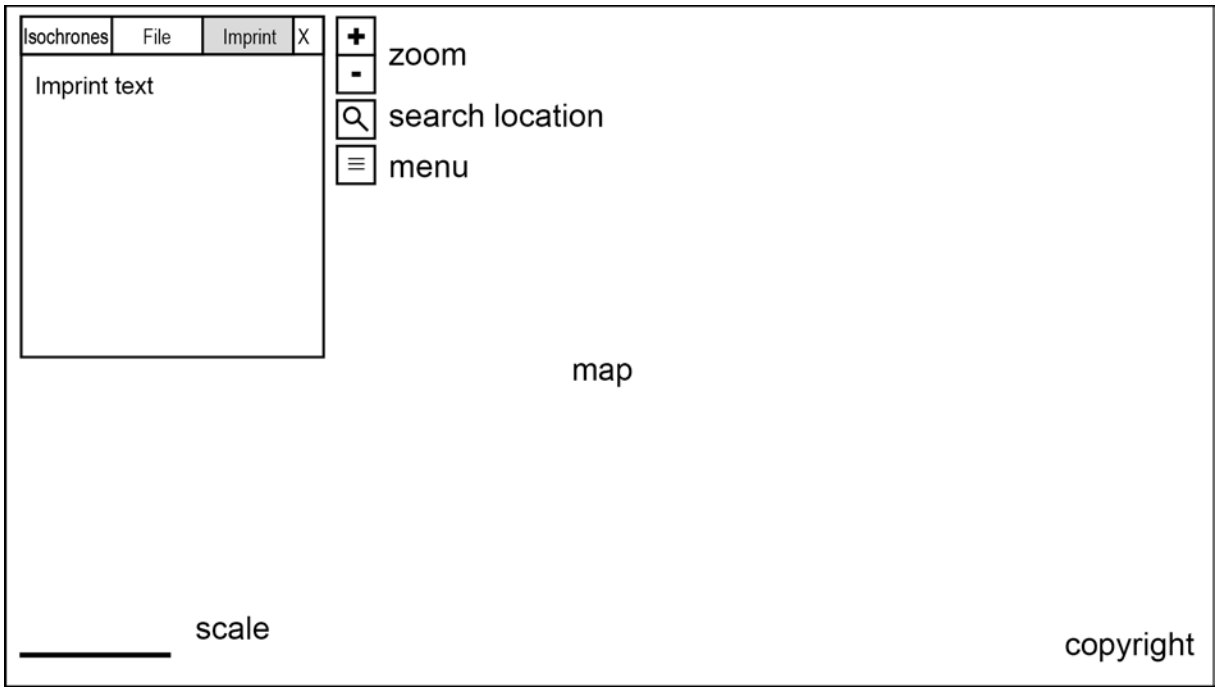

Figure 34: The Imprint tab shows information about the imprint.

6.3 Profiles

"A routing profile is a set of rules that a routing engine [...] uses to find the optimal route between two points. Routing profiles are generally optimized for the mode of transportation being used to get between locations."[142]

Emergency services use numerous different vehicles on land, at sea, and in the air to rescue people or to avoid damage to property or environment. Subsequent only land vehicles are treated. Most of them correspond to a standard. A standard ensure a minimum range of equipment that is carried on a vehicle with this standard to enable the vehicles to handle specific operations. This is important because a public-safety answering point needs to know immediately which vehicle can be sent to a special emergency and which not. These standards vary between countries, but neighboring countries often have similar standards. For example 34 European countries together develop European Standards (ENs) in the European Committee for Standardization (CEN).[143] Also some other countries oriented their standards to these European Standards.[144] European Standards are adapted to national standards and therefore named with the abbreviation for national standards and for European Standards, for example DIN EN in Germany.[145] One of these standards is the DIN EN 1846, a standard which defines all different types of fire trucks. Therefore fire trucks are very similar in all European countries. In this standard the vehicles of the fire brigade are classified into three categories according to their weight. These categories are light (L: 3-7.5t), medium (M: >7.5t up to 16t), and heavy (H: heavier than 16t).[146] In addition to that the DIN Standards Committee Firefighting and Fire Protection (DIN-FNFW) divides the three categories further in the six classes L I (3-4.75t), L II (> 4.75 up to 7.5t), M I (> 7.5 up to 9t), M II (> 9 up to 14t), MIII (> 14 up to 16t), and H (heavier than

[142] MAPBOX, 2017
[143] EUROPEAN COMITTEE FOR STANDARDIZATION: CEN MEMBERS, 2017
[144] EUROPEAN COMITTEE FOR STANDARDIZATION: CEN AFFILIATES, 2017; EUROPEAN COMITTEE FOR STANDARDIZATION: LIST OF COMPANIONS, 2017
[145] DIN DEUTSCHES INSTITUT FÜR NORMUNG E.V., 2017
[146] STAATLICHE FEUERWEHRSCHULE WÜRZBURG, 2016, p. 7

16t) in DIN SPEC 14502-1. In their type list of the most common standardized fire trucks they also summarize the vehicle dimensions and weight of 24 different types of fire trucks.[147]

The profiles for the accessibility analysis are formed according to the six classes of DIN SPEC 14502-1. An additional profile for ladder trucks is formed, because of their much larger vehicle dimensions. The vehicles of EMS and THW are compared with these profiles and assigned to the corresponding one. In some cases the profiles have to changed a bit. EMS vehicles are standardized in DIN EN 1789, to develop the profiles also the specifications of the Bavarian Red Cross (BRK) for the different kinds of EMS vehicles were used.[148] Vehicles of THW are assigned to the profiles according to the list of all vehicle types of the THW federal association, information about the vehicles of chapters on their websites, and specifications about the different divisions of the THW (StANs).[149]

All profiles have some emergency attributes in common which enables vehicles of these profiles to use among others roads that were built specifically for emergency services like emergency accesses to motorways. Moreover emergency service vehicles have the permission to drive faster than the permitted maximum speed and to ignore traffic rules in consideration of that they do not endanger or force other road users if they use special signs.

The maximum vehicle dimensions, weight, and speed of the profiles are defined as shown in table 1. The values are oriented towards the previously mentioned standards and regulations as well as an average value of many different vehicles for the maximum speed of category LI.

Category	LI	LII	MI	MII	MIII	H	Ladder
Max. weight	4.75t	7.5t	9t	14t	16t	-	16t
Max. length	6.5m	8.0m	8.0m	8.6m	10m	10m	11m
Max. width	2.3m	2.55m	2.55m	2.55m	2.55m	2.55m	2.5m
Max. height	3.1m	3.3m	3.4m	3.5m	3.5m	4m	3.3m
Max. speed	120km/h	100km/h	100km/h	100km/h	100km/h	100km/h	100km/h

Table 1: Definition of the profiles.

In addition to the maximum speed shown in Table 1 it is also necessary to define different speeds for different road types. Beside the in chapter 2.5.1 mentioned average speeds in town and out of town no more precise information is available at the State Fire Brigade School Baden-Württemberg[150] and at the volunteer fire brigade of Constance.[151] Also career fire brigades do not have much more information. For example the career fire brigade of Wiesbaden performed test drives to compute their accessible area[107] and the career fire brigade of Karlsruhe analyzed measured emergency drives.[152] For this reason the speed for different road types in case of an emergency drive is defined as 10% faster than the permitted maximum speed.

[147] DIN-NORMENAUSSCHUSS FEUERWEHRWESEN, 2016
[148] BAYRISCHES ROTES KREUZ, 2017
[149] BUNDESANSTALT TECHNISCHES HILFSWERK, 2017
[150] HÜSCH, 2017
[151] FISCHER T. , 2017
[152] WOLTER, 2017

To check the response time required by law only the inspection for the slowest vehicle is necessary, because the response time is only fulfilled if all needed vehicles are at the scene. But there are also laws for the maximum response time of some special vehicles. To check them there are a few additional profiles. The most important and most used vehicles of the three analyzed emergency services are assigned to the different profiles in table 2. In addition to that it is not reasonable to create an own profile for each type of emergency vehicle because there is a wide range of different vehicle types. There are also some special vehicles which do not meet the standard for any vehicle type.

Class	Examples
LI	Command Vehicle Emergency Ambulance Emergency Physician Car Patient Transport Ambulance Personnel Carriers Small Command Support Unit Small Fire Engine Squad Truck type 1 Squad Truck type 5
LII	Mobile Intensive Care Unit Small Operational Support Unit Squad Truck type 2
MI	Medium Fire Engine
MII	Equipment Vehicle type 2 Fire Engine HazMat (Hazardous Materials) vehicle Heavy Rescue Vehicle Small Water Tender Squad Truck type 3 Squad Truck type 4
MIII	Large Command Support Unit Large Fire Engine Large Operational Support Unit Large Water Tender
H	Equipment Vehicle type 1 Multi-Purpose Vehicle Swap Body Vehicle
Ladder	Aerial Ladder Platform Turntable Ladder

Table 2: Examples of the most important vehicles of fire brigades, EMS, and THW inclusive their assignment to the previous defined profiles.

6.4 Test and selection of the routing algorithm

Three of the four routing algorithms mentioned in chapter 5.2.2 are investigated. OpenRouteService is not useable because of its usage restrictions.

The main characteristics of the remaining three routing services are summarized in the following table.

Service	GraphHopper	OSRM	Valhalla
Open source base-version	yes	yes	yes
Complete open source version (especially isochrone part)	no	yes	yes
Isochrone	Isochrone API	Extension OSRM-isochrone	yes
Algorithm	selectable between CH, A* and Dijkstra	CH	A* with improvements
Language	Java	C++	C++

Table 3:Comparison between GraphHopper, OSRM, and Valhalla. The abbreviation CH stands for Contraction Hierarchies.

All of the presented routing services are at least partly open source and use OpenStreetMap data for their routing graph. Therefore all of them are usable worldwide.

The Isochrone API of GraphHopper is closed source and just usable with an API key, hence this service is also not usable, because it does not confirm to the guidelines defined in chapter 1.2.

In principle OSRM-isochrone as well as Valhalla confirm to all guidelines and are consequently useable, but as already mentioned the support of OSRM-isochrone is not as good as for Valhalla.

As a result Valhalla is the best choice for this task.

7 Implementation

The development part of this work focuses on a user-friendly frontend for accessibility analyses. The computation backend makes use of open source routing algorithms. As described in chapter 6.4 Valhalla is chosen as routing algorithm for this project. Only some small adaptations have to be done at this point as discussed in chapter 7.3.
The frontend is coded in HTML, CSS, and JavaScript. HTML is used to define the structure of the website, the design is done by CSS, and JavaScript is used to program the interactivity like sending routing requests as well as receive, interpret and present the response from the routing algorithm at the backend.

For clarity the HTML, CSS and JavaScript code parts are divided into different files. Different files are also used for different functions or groups of functions. The folder structure is clearly divided, too. The Valhalla backend is stored in the folder called "valhalla" and the frontend is stored in the folder called "frontend". Inside the frontend folder there are two folders, one for all code files and one for the image files. The code for the website of this project can be found at GitHub (https://github.com/BFutterer/Masterthesis).

Figure 35 shows the architecture of this project. The part inside the gray box is running on the development machine and the other parts are used as services.
All functionalities are controllable via the website, therefore it uses the libraries Leaflet, Turf.js, and jQuery besides custom functionalities. Leaflet is used to display the map tiles of the OpenStreetMap tile server as well as the computed isochrones, the markers for the starting locations, and the boundaries at the website. Turf.js is needed to merge the GeoJSON files and jQuery allows to simplify and reduce the code. Accessibility analyses are performed by a local installation of the routing engine Valhalla.
The OpenStreetMap tile server provides the map tiles that are embedded into the website with the help of the Leaflet library, and the Nominatim service enables the geocoding functionality as well as the search for the boundaries of cities.
The most important parts of the code used for this project are explained in the following chapters.

To develop this project only a routing graph for Switzerland and Liechtenstein is generated. This is also possible for all other countries. However the generation of a worldwide routing graph needs a lot of time.

Moreover accessibility analyses should be possible worldwide. In this project this is only possible for.

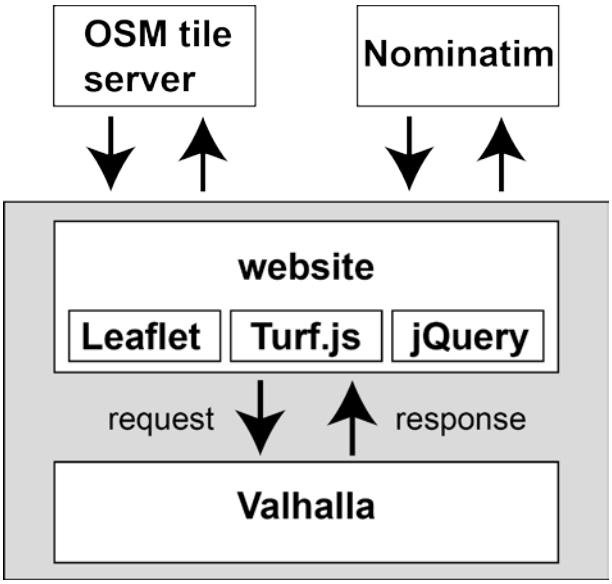

Figure 35: The architecture of the project. The gray box defines the parts of the project that are running on the local machine. The key component is the website that allows to control the accessibility analysis. Additional functionalities are imported to the website by adding the libraries Leaflet, Turf.js, and jQuery. To perform the accessibility analyses a copy of the routing engine Valhalla is set up locally (see chapter 7.2). OSM map tiles (background map, see chapter 7.2.1) and Nominatim (geocoding and boundaries for the inspection function, see chapters 7.4.3 and 7.4.4) are used as services via an external interface.

Accessibility Analysis is set as title of the website and also an appropriate favicon is added to be visualized in the tab for the website. This can be seen in figure 36.

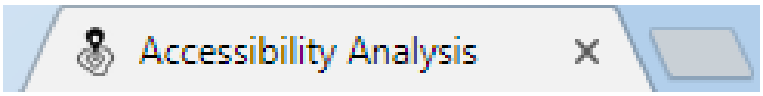

Figure 36: Favicon and title of the website are displayed in the tab.

7.1 Graphical User Interface

The Graphical User Interface (GUI) is used to enable a user-friendly handling of the accessibility analysis. Via the GUI the user can control all important settings to adapt the analysis to their needs. Moreover the user is able to store, import and merge files via the GUI. Figure 37 shows the default display of the GUI.

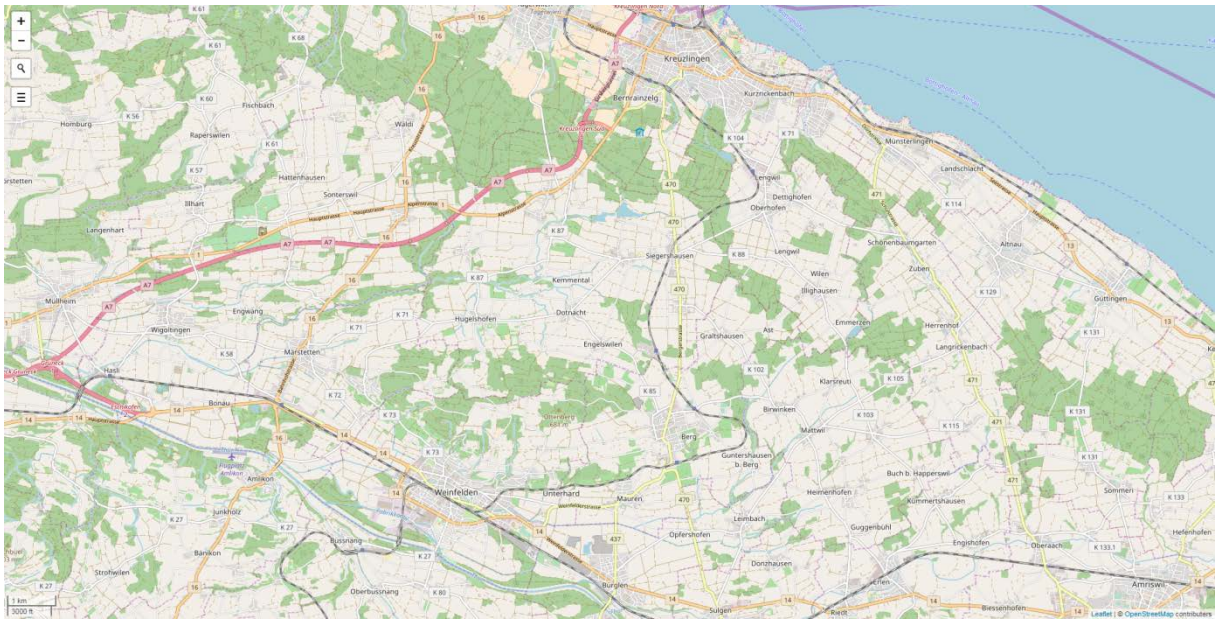

Figure 37: The GUI of the project. The map covers the complete screen. In the upper left corner of the screen controls for zooming, search for locations, and open the settings menu are placed. The scale bar can be seen in the lower left corner.

7.1.1 Layout

To ensure a good visibility of the result of the accessibility analysis the map covers the complete screen. Some basic functions can be chosen by selecting one of the buttons in the upper left corner. Additional settings can also be achieved via this buttons.

Icons are used instead of text on the buttons to reduce their size as well as to simplify the usage. A tooltip briefly explains the respective function of each button if it is not clear. The tooltips can be opened by a mouse over, this can be seen in figure 38. Equally the functions inside the settings menu are explained with the help of tooltips as figure 39 shows.

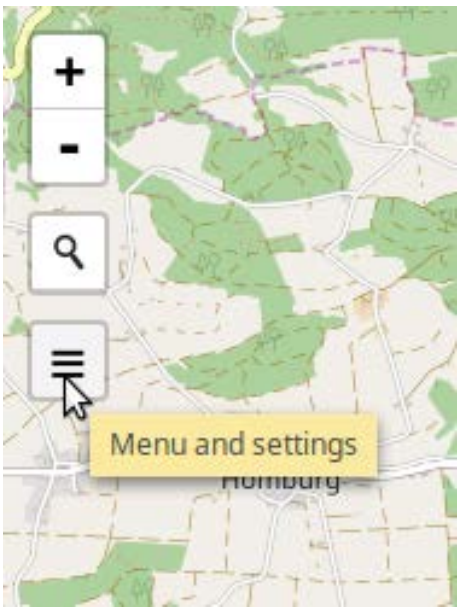

Figure 38: A tooltip explains the functionality of each button if it is unclear. It can be opened by a mouse over.

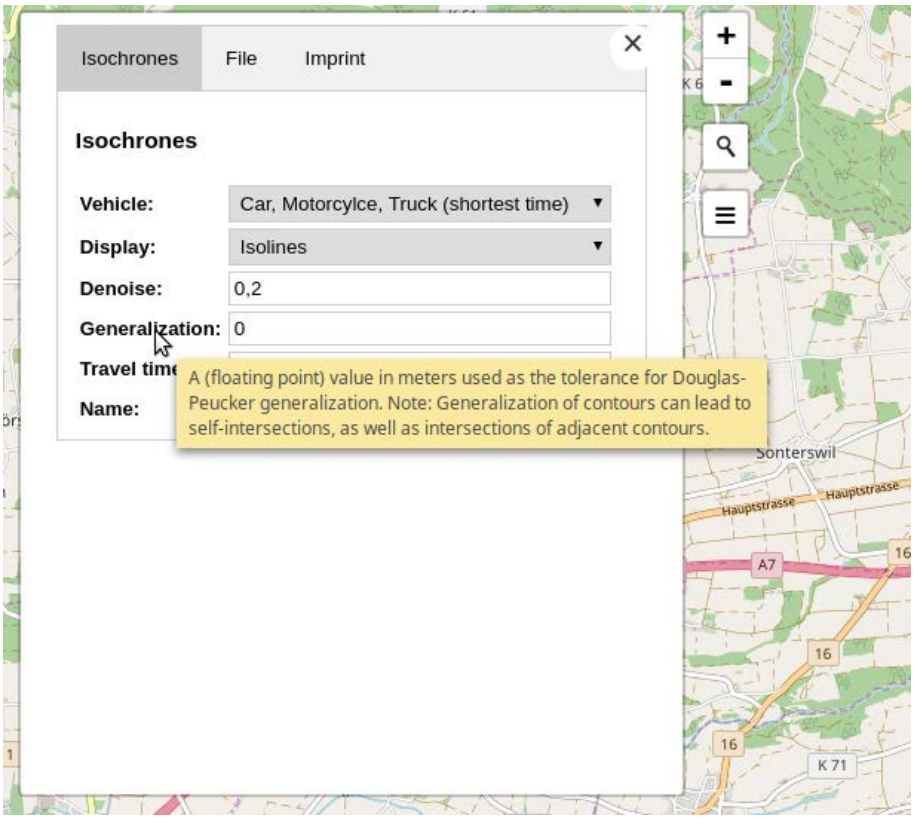

Figure 39: The functions inside the settings menu are also described by a tooltip.

The upper two buttons can be used for zooming. Alternatively zooming also can be done using the mouse wheel.

The magnifier represents a locate function with which the user is able to search for a desired location using Nominatim, the center of the map will automatically zoom to this location. If the button has been selected a search filed like it is visible in figure 40 appears.

A click on the lowest buttons opens the complete settings menu. This menu is divided into three tabs. The first is used to adapt the accessibility analysis to the specific needs. Therefore for instance the vehicle or the travel time can be set here (see figure 41).In the second tab it is possible to reset the map to the default settings as well as export, import, and merge files. The user can visually check whether all necessary areas can be reached within the predefined time period. Figure 42 shows the second tab.

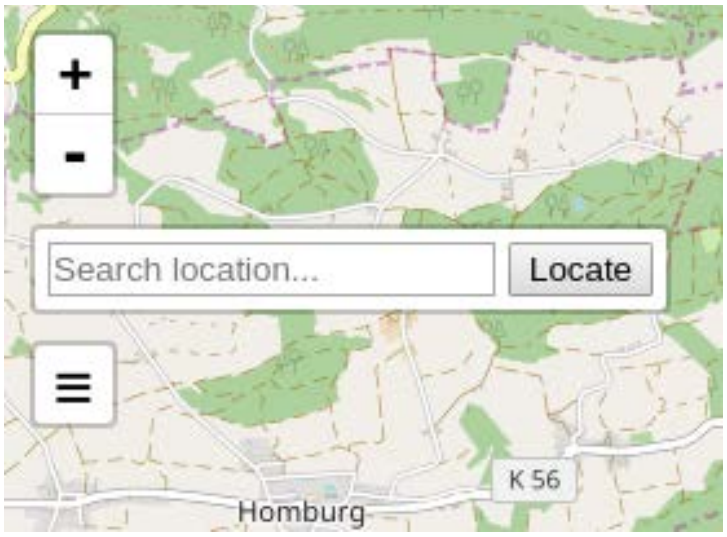

Figure 40: Selecting the magnifier button opens the search location function. A desired location can be written in the text field to look for it. The center of the map will automatically zoom to this location.

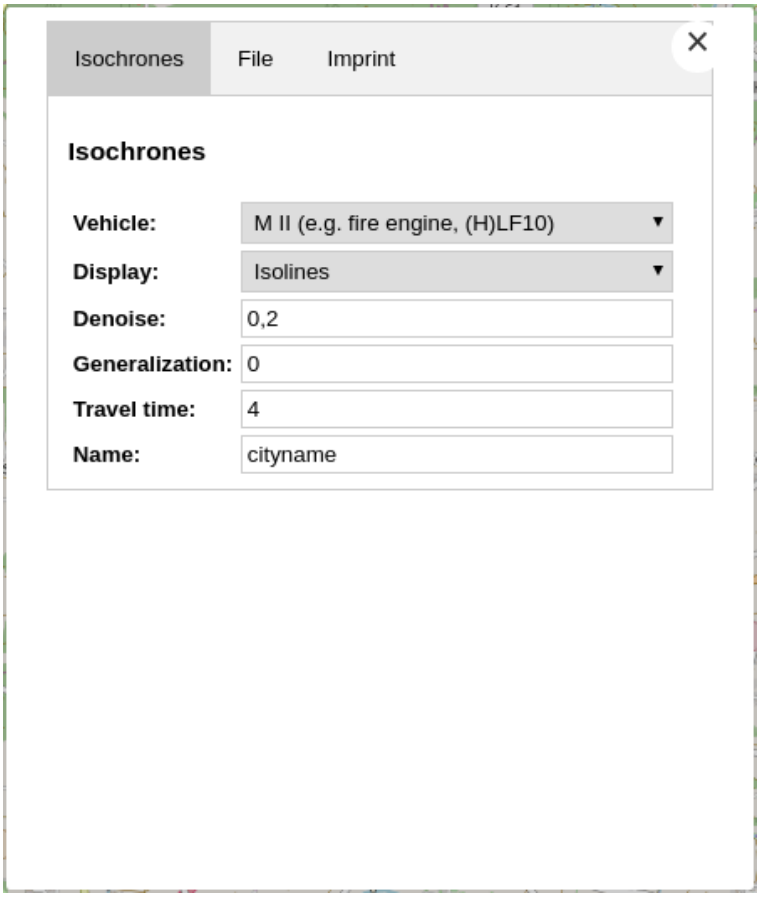

Figure 41: Available settings for adapting the isochrone computation. The profiles are named according to classes of DIN SPEC 14502-1 (see chapter 6.3). To make the selection easier for the user an example is given for each class.

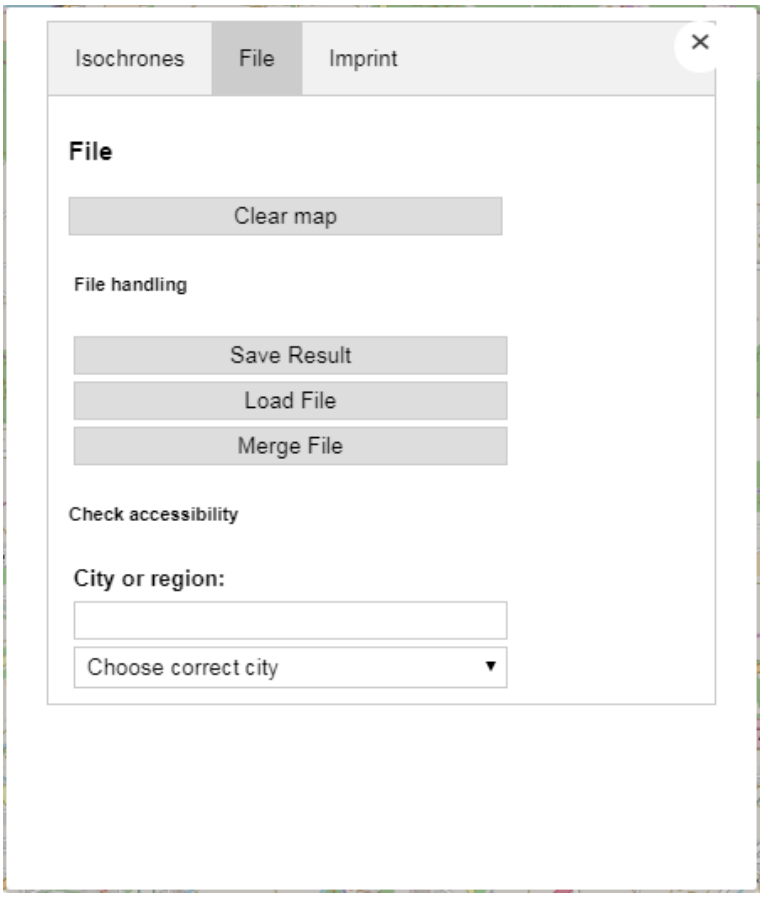

Figure 42: Settings to reset the map, handle files and check if all necessary areas are reachable within the predefined time period.

7.1.2 Menus

The settings menu can be fade in by selecting the menu button. If the button is selected again or the close button in the settings menu is clicked the settings menu is faded out. As described in the previous chapter the settings menu is divided into three tabs. These tabs are defined in a class called "tab" like it is visible in figure 43. The functionality of these tabs is coded in the JavaScript file sidebar.js.

```
<div class="tab">
    <button class="tablinks active" onclick="openTab(event, 'Isochrones')" id="defaultOpen">Isochrones</button>
    <button class="tablinks" onclick="openTab(event, 'File')">File</button>
    <button class="tablinks" onclick="openTab(event, 'Imprint')">Imprint</button>
</div>
```

Figure 43: Definition of the tabs for isochrone computation, file handling and imprint. The tab for isochrone computation is by default set as active tab.

A table is used to define the structure of the first two tabs. These tables are enclosed by a form element to enable sending the chosen settings to a server.
Inside the table the different settings options are programmed via various HTML elements. For drop-down menus the select element is used. Figure 44 shows the implementation of the vehicle selection. In addition to that drop-down menus are also used for defining the display of the result (as isochrones or as polygon) and for choosing the correct city from a list of results after search.

```
<select id= "select_vehicle" name="vehicle" size="1" onchange=" generateIsochrones()">
    <option value="auto">L I (e.g. small fire engine, TSF)</option>
    <option value="auto">L II (e.g. small operational support unit, GW-L1)</option>
    <option value="truck">M I (e.g. medium fire engine, MLF)</option>
    <option value="truck" selected="selected">M II (e.g. fire engine, (H)LF10)</option>
    <option value="truck">M III (e.g. large fire engine, (H)LF20)</option>
    <option value="truck">H (e.g. swap body vehicle, WLF)</option>
    <option value="truck">Ladder (e.g. turntable ladder, DLK 23/12)</option>
</select>
```

Figure 44: Definition of the drop-down menu for choosing the desired vehicle via a select element. If the vehicle has changed the isochrone computation is repeated (function generateIsochrones()). For an explanation of the profiles see chapter 7.3.

Input fields are used to enter text and to change the default values of numbers or to display buttons. Depending on the purpose for which the input is used input fields of the type button, number or text are used. Input fields of type button create selectable buttons.

The type number prevents the input of other characters than numbers, whereas all characters can be inserted into input fields of type text. If the type number of input fields is used it is also possible to define a minimum and a maximum value that can be entered as well as a default value. The code for the input field for the travel time is visible in figure 45. Beside selecting travel time input fields of type number are used to define the denoise and the generalization value. Input fields of type text are used for enter a name for the request and the city name to check if all necessary areas are reachable within the set time period.

```
<input id="minutes_value" type="number" min="1" max="30" step="1" value="4" onchange="minutes = this.value, generateIsochrones()">
```

Figure 45: Definition of the input field for the travel time. It is possible to insert numbers (type) between 1 (min) and 20 (max). The default value is 4 (value). If the value has changed the isochrone computation is repeated.

Buttons are used to create the tab chooser (see figure 43) as well as to offer the user the opportunity to cause the actions reset, save, import, and merge. If a button is clicked the respective defined click event is triggered (onclick). Figure 46 shows the definition of the buttons for export a file and import one file or several files.

```
<tr>
    <td>
        <!--Export-Button-->
        <a href="#" id="export"><button type="button" title="Save result to a geoJSON file">Save Result</button></a>
    </td>
</tr>
<tr>
    <td>
        <!--Import-Button-->
        <input type="file" id="import" name="files[]" multiple accept=".geojson" method="post" enctype="multipart/form-data" style="display:none;"/>
        <input type="button" value="Load File" title="Load one or several geoJSON files to the map" onclick="document.getElementById('import').click();" />
        <script>
            document.getElementById('import').addEventListener('change', handleFileSelect, false);
        </script>
    </td>
</tr>
```

Figure 46: Definition of the export and the import button. Visible is also a part of the table structure. Each button is located in an own table row (tr element) and an own column (td element). The path to the location where the export file should be stored replaces the variable "#" in the link attribute (href) later on. Clicking the import button opens a file chooser (type = file) that only displays GeoJSON files (accept=".geojson"). The selection of several files is possible (multiple). Because HTML 5 only allows to open a file chooser via an input field of type file, but due to an uniform appearance an additional button is needed and the input field is set invisible (style="display:none;"). If the button is clicked it triggers the click event of the input field.

7.2 Integration of Valhalla

For developing this project version 2.3.0 as from 21 July 2017 is used as backend.[139] The project development was done on a Linux operating system.

There are installation instructions at the GitHub repository of Valhalla (https://github.com/valhalla/valhalla). Either Valhalla can be built from source or a Personal Package Archive (PPA) can be used. Installing Valhalla via PPA has the advantage that it is quicker and easier than building from source. Therefore this way is used in this project.

At first the Valhalla GitHub repository has to be cloned. This takes around one minute. Although this is normally not needed if PPA is used, it was done this way in this work.
Before starting to install Valhalla changes in the source code of Valhalla can be made. In this case the number of isochrones has been increased to 30 in the valhalla_build_config file (see chapter 7.2.2.2 for further information).
Subsequently the PPA can be installed like described in the "Get Valhalla from Personal Package Archive (PPA)" paragraph, which takes another minute.
Afterwards the commands below the hint "After getting the dependencies install it with" have to be executed. Therefore circa 10 minutes are required. This step has also to be executed each time something has been changed in the Valhalla routing algorithm.
Finally the OSM data have to be downloaded and the routing tiles have to be computed out of this OSM data. This is described in the "Running" section. Daily updated OSM data for the desired countries or regions can be downloaded as data.osm.pbf from the website of the Geofabrik (http://download.geofabrik.de).[153] For Switzerland and Liechtenstein this took approximately nine minutes. This paragraph describes also how to start the server and open an example website.
To start the server again later on, each time the command `valhalla_route_service valhalla.json 1` has to be used.

The previously mentioned durations for the separate installation steps depends on the machine and internet connection. They should only be interpreted as a reference value.

7.2.1 Primitive map

The installed Valhalla code already includes an example file (see https://github.com/valhalla/demos/blob/gh-pages/isochrone/index.html) that can be modified and extended. A detailed description how to add Valhalla to a website can be found at https://github.com/valhalla/valhalla-docs/blob/master/add-routing-to-a-map.md. The Leaflet library is used to display a map in the website and therefore has to be imported.

The HTML code has to contain a div element with the ID map and a size lager than zero in which the map will be displayed. This looks for example similar to this: `<div id="map" style="width: 1440px; height: 960px"></div>`. However the size can also be set in a separate CSS file instead of using the style attribute. In addition to that the map is placed into this map element by `map = L.map('map').setView([47.6, 9.15], 13);`, whereat the pair of coordinates sets the center of the map and the single number defines the zoom level of the map. The higher the zoom level the larger the scale and more detail the map. The L.-statement is a reference to Leaflet. To show the OSM map in the map element also the code in figure 47 has to be added.

[153] GEOFABRIK, 2016

This is already enough to add a OSM map via Leaflet to a website.

```
L.tileLayer('http://b.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  maxZoom: 18,
  attribution: '&copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributers'
}).addTo(map);
```

Figure 47: Code snippet to add the OSM map to the map element. Moreover the maximum zoom level is set to 18 and a copyright is defined.

7.2.2 Settings

Valhalla offers a lot of settings to influence the isochrone computation in the range from fundamental settings (like the maximum number of isochrones or the maximum travel time) to special settings such as generalization of the result or selection of a routing profile. These settings are explained in more detail in the next two chapters.

7.2.2.1 Default settings

By default Valhalla computes a maximum of four isochrones in the range of 120 minutes or 25 kilometers. The isochrones are displayed in default colors, if no other colors are specified. It is only possible to define one start location.[154] In addition to that Valhalla sends isochrone computation requests by default to Mapzen's Time-Distance-Matrix service and an API key is needed for that. Computations are performed for the car profile. The response is visualized as isochrones with an generalization of 150 meters in the Douglas-Peucker algorithm. All areas that are smaller than 20% of the main area of the same time period are dropped. The starting location is defined by clicking into the map.[155]

7.2.2.2 Adapted settings

Some of the settings described in the previous chapter have to be increased and other settings have to be made changeable by the user to give them the possibility to adapt the isochrone computation to their needs.

Above all it is important to increase the number of isochrones, because for each minute an isochrone should be drawn. Breathing apparatus units and HazMat vehicles have a maximum 30 minutes travel time, because they have to cover a larger area than all other types of fire trucks. Because of this it is the longest travel time of all listed fire trucks at the References to the capability of the fire brigades.[156] Therefore this value is defined as maximum travel time and therefore the maximum time for that isochrones have to be computed. Additionally the color of the isochrones or polygons is defined in a color gradient from green to red depending on their time value. Green is used for short travel times and red is used for long travel times. The computation of the color of each time step is done in the HSV color scheme, because it is easier to compute the values just for one scale (Hue-scale 0 to 1) instead of interdependent 3 scales (red, green, blue). But the RGB color scheme is necessary to be able to display the colors on the screen. Therefore the HSV have to be converted into RGB values. This is done by the function HSVtoRGB that is visible in figure 48. Figure 49 shows the code to compute the appropriate color for each time step. Each isochrone is defined by a statement such as {"time":20,"color":"#abcdef"}.

[154] DARA-ABRAMS, et al., 2017
[155] KREISER, Isochrone demo, 2016
[156] LANDESFEUERWEHRVERBAND BADEN-WÜRTTEMBERG, 2008, pp. 20-21

```
function HSVtoRGB(h, s, v) {
  var r, g, b, i, f, p, q, t;
  if (arguments.length === 1) {
    s = h.s, v = h.v, h = h.h;
  }
  i = Math.floor(h * 6);
  f = h * 6 - i;
  p = v * (1 - s);
  q = v * (1 - f * s);
  t = v * (1 - (1 - f) * s);
  switch (i % 6) {
    case 0: r = v, g = t, b = p; break;
    case 1: r = q, g = v, b = p; break;
    case 2: r = p, g = v, b = t; break;
    case 3: r = p, g = q, b = v; break;
    case 4: r = t, g = p, b = v; break;
    case 5: r = v, g = p, b = q; break;
  }
  return {
    r: Math.round(r * 255),
    g: Math.round(g * 255),
    b: Math.round(b * 255)
  };
}

function toPaddedHexString(num, len) {
    str = num.toString(16);
    return "0".repeat(len - str.length) + str;
}
```

Figure 48: Function HSVtoRGB that is used to convert the HSV values of each isochrone to RGB values to be able to display them on the screen. In addition to that the function toPaddedHexString is used to add a 0 in front of hexadecimal numbers. This is necessary because Valhalla needs colors in a scheme like #abcdef". These functions can be found in the map.js file.

```
for(i=1; i <= localminutes; i++) {
  var color = HSVtoRGB((96/255)*(localminutes-i)/(localminutes), 1, 1);
  contour[i - 1] = {"time": i,"color": toPaddedHexString(color.r, 2)+toPaddedHexString(color.g, 2)+toPaddedHexString(color.b, 2)};
}
```

Figure 49: Function to compute the color of each individual isochrone. The final definition of an isochrone with appropriate color looks like {"time":20,"color":"#abcdef"}. This code snippet can be found at the function generateIsochrones in the map.js file.

The maximum number of isochrones can be changed in the valhalla_build_config file in the scripts directory. At line number 189 the value 4 has to be replaced by 30. Figure 50 shows the mentioned code part of the original code.

In addition to that the URL to which the request should be send is changed from Mapzen's Time-Distance-Matrix service (`https://matrix.mapzen.com/isochrone?json=`) to the local host (`http://localhost:8002/isochrone?json=`). This can be found in the map.js file of the project at line 212.

To allow a more exact inspection of the result, the maximum zoom level of the embedded OSM map is increased to 19 in the bodyLoadScript.js file, which is also the maximum zoom level of OSM itself.

The user should be able to adapt the following settings to their needs. These settings cover the choice of a profile, the presentation method for the result (isochrone or polygon), the generalization of the result as well as the definition of a denoise value to drop smaller areas, the determination of the travel time (the computation of all possible isochrones is not always reasonable) and an optionally definition of a name for the computation. The implementation of this is already described in chapter 7.1.2. Figure 41 shows the coded default values for each of the mentioned settings.

For all other settings the default settings are kept.

```
188       'isochrone': {
189         'max_contours': 4,
190         'max_time': 120,
191         'max_distance': 25000.0,
192         'max_locations': 1
193       },
```

Figure 50: Code snippet that have to be changed in the valhalla_build_config file. The variable max_contours defines the maximum number of isochrones.

7.3 Vehicle Profiles

Valhalla allows the use of different routing profiles for the computation. Several routing profiles for different modes of transportation like car, bicycle, and pedestrian are available by default. An overview of many available profiles can be found on the website of Mapzen at https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/#costing-options. It is also possible to add custom profiles by deriving them from existing profiles. The profiles are called costing methods and are stored in so called "costing files". With the aid of these costing files Valhalla is able to generate dynamic and run-time costing, which means the costs are not directly added to the routing graph but stored separately. Therefore the graph does not have to be recalculated to add a new profile. Costing computation is done in a module called "Sif".

To add a new profile it can be derived in an already existing costing file or a new costing file for the profile can be created. A new profile typically requires a source and header file (costfile.cc and costfile.h). Both have to be added to Makefile.am for the build process. Deriving the profile inside an existing file does not require any new files.

The new profile has also to be added to the worker.cc files of the modules Thor and Loki as valid costing option to get added to the CostFactory function. This function computes the costs for the edges dependent on the profile. In the file valhalla_build_config the maximum distance of a route and the maximum number of locations visited per route can be defined. After a new profile has been added, Valhalla has to be compiled again to offer this profile for computing the isochrones.[157]

For the emergency service profiles the attributes for maximum speed, vehicle dimensions and weight are necessary. These attributes differ between the classes defined in chapter 6.3, but all of them

[157] KREISER & NESBITT, SIF - Dynamic Costing within Valhalla, 2017; NESBITT D. , How to add new profiles to valhalla?, 2017

have the permission to drive on all roads in common. For this reason it is useful to create a new costing file that contains all the profiles for each class.

It was attempted using both of these methods to create a costing for specific emergency service profiles, but due to lack of accessible examples or documentation this turned out to be too time consuming for this thesis.

While the vehicle characteristics of emergency services vehicles differ from those of regular vehicles, the difference in the resulting isochrones is not expected to be significant and hence it was decided to work with the default auto and truck profiles. Therefore these profiles have been assigned to the classes as can be seen in table 4.

Class	Used Valhalla profile
LI	auto
LII	auto
MI	truck
MII	truck
MIII	truck
H	truck
Ladder	truck

Table 4:Used default profiles of Valhalla for each class.

7.4 Coded functionality

Beside the computation of the isochrones it is also mandatory to offer functions to the user to enable them to store the result as well as to import results they have already computed. As mentioned in chapter 6.1 the possibility to define several starting locations is waived in favor of the implementation of a merge function, because this allows also the combination of the results of accessibility analyses with different settings. In addition to that it is reasonable to compare the result of the accessibility analysis with the boundary of the alarm response area respective the city to find areas that are not reachable in the prescribed time interval. Additionally a geocoding function is included to locate the desired city. All of these functions are described below.

7.4.1 Isochrone computation

The user can influence the computation by adapting the computation attributes to their needs. As shown in figure 41 the five attributes vehicle (for computation used routing profile), display (of the result), denoise (dropping of small areas), generalization (of the result), and the travel time (maximum time for that isochrone should be calculated) can be changed. These attributes are customized via the Isochrone tab of the settings menu. The user has the possibility to add a name to the request in order to better differentiate between several accessibility analyses. Last but not least the starting location is defined by clicking on the desired place in the map. If one of the mentioned attributes has changed the isochrones are recomputed. This is done by calling the function generateIsochrones in the map.js file. Figure 51 shows the part of this function in which the URL is build to send a request to the Valhalla backend.

At the beginning of this function it is checked if a starting location has been chosen. If not nothing more is done, else the next commands are executed. To request a single isochrone the syntax: `{"time":20,"color":"#abcdef"}` is needed. The definitions of all isochrones are collected in the array contour. This requires the value of the currently chosen travel time and is done like already described in chapter 7.2.2.2 and visible in figure 48 and figure 49. Next the adapted values of all attributes are attached to the fixed front part of the request URL. The command

`encodeURIComponent` is used to enable also special characters for the name. This encodes special characters to their corresponding ASCII value for the request, for example an ampersand character ("&") is replaced by %26.

```
//generate the isochrones
function generateIsochrones() {
  //check if a startLocation is set, if not return
  if(typeof startLocation == 'undefined') return;
  //draw as many isochrones as minutes defined for travel time and set their color automatically from green to red
  var contour = new Array();
  var localminutes = $("#minutes_value").val();
  var localsetId = $("#giveId").val();
  for(i=1; i <= localminutes; i++) {
    var color = HSVtoRGB((96/255)*(localminutes-i)/(localminutes), 1, 1);
    contour[i - 1] = {"time": i,"color": toPaddedHexString(color.r, 2)+toPaddedHexString(color.g, 2)+toPaddedHexString(color.b, 2)};
  }
  //build url to send to the backend for computation of the isochrones
  //fix part
  var url = 'http://localhost:8002/isochrone?json=';
  //adaptable part
  var json = {
    locations: startLocation,
    costing: $("#select_vehicle").val(),
    contours: contour,
    polygons: $("#polygon_value").val(),
    denoise: $("#denoise_value").val(),
    generalize: $("#generalization_value").val(),
    id: encodeURIComponent(localsetId)
  };
  //creation of the complete URL (url + json) and convert the URL in an appropriate format
  url += escape(JSON.stringify(json));
```

Figure 51: Code snippet of the first part of the generateIsochrones function. In this part the URL is build to send a request to Valhalla.

In the following step all potentially existing layers except the OSM layer are deleted. This is done by the code displayed in figure 52.

```
//clear old marker and other layers
map.eachLayer(function (layer) {
  if(layer != osmLayer) {
    map.removeLayer(layer);
    }
  });
```

Figure 52: Code used to delete all potentially existing marker, isochrone, and boundary layers from the map. Only the OSM base map remains unchanged.

Then the URL is sent to the backend and the inner function computeIsochrones is called. In this function all potentially existing isochrones (variable geojson)) and tooltips are deleted, before the new isochrones are added to the map by calling the addGeoJson function. Furthermore a marker is added to the map to display the clicked position. In addition to that a tooltip with the name chosen for the request and the location is appended to the marker. Figure 53 shows the code described.

```
  //grab the url
  $.getJSON(url,function computeIsochrones(isochrones){
    //clear geojson if its not null
    if(geojson != null)
      geojson.removeFrom(map);
    //clear the tooltips
    tooltips.forEach(function (tooltip) {
        tooltip.removeFrom(map);
    });
    tooltips = [];
    //create the geojson object by calling the addGeoJson function
    addGeoJson(isochrones);
    //copy the geojson response of Valhalla (called isochrones) to a global variable
    copyIsochrones = isochrones;
    copyPosition = position;
  })
  //set new marker
  //set a marker to the current clicked position
  var marker = L.marker(position);
  map.addLayer(marker);
  //display the coordinates of the clicked point in the tooltip window
  marker.bindPopup("<b>Name:</b><br />"+localsetId.toString()+"<br />
                    <b>Starting point:</b><br />" + "lat: " +
                    (Math.round(position.lat * 10000) / 10000) +", long: " +
                    (Math.round(position.lng * 10000) / 10000));
}
```

Figure 53: Code extract of the last part of the generateIsochrone function the inner function computeIsochrones. Before the new isochrones are computed the potentially existing old isochrones and tooltips are removed. Moreover a marker is set to the chosen starting location inclusive a tooltip that shows the chosen request name and the clicked position.

Valhalla computes the isochrones by dividing up the map into a two-dimensional grid and computing the cost to reach each grid square from the starting point. For that the Dijkstra algorithm is used. The cost calculation stops when the maximum time interval is reached. The resulting two-dimensional grid with travel costs to each cell from the starting location is used to compute the isochrone contours with the aid of CONREC. Because CONREC only provides separate line segments, they have to be pieced together to form closed contour lines.[136] However this raster based approach to create isochrones causes a bit of imprecision of the result.

Finally the response is visualized via the addGeoJson function by using Leaflet. This function also appends tooltips to the isochrones that indicate the respective time interval. Figure 54 shows only the basic part of the code for this function because of clarity. The remaining part is explained in chapter 7.4.2.3. In Figure 55 the result of an isochrone computation for the fire station Luzern Kleinmatt can be seen. This figure also shows the tooltip that is opened if the user selected the marker of the starting location. Inside this tooltip the inserted name and the location of the marker are visible.

```
//add isochrones to map
function addGeoJson(json) {
  geojson = L.geoJson(json, {
    style: function(feature) {
      return {
        opacity: feature.properties.opacity * 2,
        weight: 10,
        color: feature.properties.color,
        //add Layer to the isochronePane to be able to define a higher z-index
        pane: 'isochronePane'
      };
    },
    onEachFeature: function(feature, layer) {
      //create a single tooltip
      var tooltip = layer.bindTooltip(feature.properties.contour + ' min', { sticky: true });
      //add the individual tooltips to the array tooltips
      tooltips.push(tooltip);
      //add the tooltips to the map
      tooltip.addTo(map);
    }
  });
  //render the geojson (creation of the picture data)
  geojson.addTo(map);
}
```

Figure 54: Basic code of the addGeoJson function. In this function the response is visualized in the map. In chapter 7.4.2.3 the extension for converting FeatureCollections to Features is described. It is added at the beginning of this function, but left in this picture because of clarity.

Figure 55: Result of the isochrone computation for the fire station Luzern Kleinmatt. Selecting the marker of the staring location opens a tooltip with the inserted name and the chosen location.

7.4.2 File handling

The complete file handling functionality is collected into the file_handling.js file. This includes an export functionality of the response (isochrones and starting point) as well as an import functionality which allows the user to import a GeoJSON file they created earlier, and a merge functionality to merge several GeoJSON files into a single one.

7.4.2.1 Export to a GeoJSON file

To save the result of an accessibility analysis into a GeoJSON file the export function is needed. At first this function checks whether the variables of the starting point as well as the isochrones are defined and therefore contain some content. If this is not the case the function issues an alert and terminates, else the content of the variables of the starting point and the isochrones are combined into a single one. Additionally headings for the two different parts are included in front of them. Before it is possible to save the content to a GeoJSON file it is necessary to convert the output into a JSON string. In addition to that the name of the output file is created from the fixed part "accessibility_analysis_" and the attached name the user inserted into the name input field ("giveId"). The extension of the GeoJSON format completes the file name. If the user inserted no name, the file name would only be accessibility_analysis.geojson. Figure 56 shows the code of the export function.

```
//Export of isochrones and start point to a GeoJSON file
function exportFunction(e) {
  //check of File APIs
  checkOfFileAPI();
  //check if the export is empty
  if (copyPosition == null || copyIsochrones == null){
    alert("No start location is set or no isochrones were computed!");
    //leave the function
    return;
  }
  //Combine GeoJSON from start point and isochrones into a single GeoJSON
  var combineExports = [{
    startpoint: copyPosition,
    isochrones: copyIsochrones,
  }]
  //stringify the combined GeoJSON to be able to save it into a GeoJSON file
  var completeAnalysis = JSON.stringify(combineExports);
  // Create export
  $("#export").attr("href", 'data: text/json;charset=utf-8,' + encodeURIComponent(completeAnalysis));
  $("#export").attr('download','accessability_analysis_' + $("#giveId").val() + '.geojson');
}
```

Figure 56: Code of the export function. The if...else statement avoids saving empty files by checking whether neither the variable for the starting point nor the variable for the isochrones is undefined. To facilitate the comprehension of the exported file a heading is added in front of the coordinates of the starting point respective the isochrones. Furthermore the output is converted to a JSON string to be able to save it in a file. Finally the file is stored in the download directory of the user the file name accessibility_analysis_CITYNAME.geojson, whereat CITYNAME represents the term the user has inserted in the name input field.

7.4.2.2 Import of GeoJSON files

With the aid of a file chooser (see figure 57) the user is able to select the GeoJSON files that they want to visualize. It is possible to select one or several GEOJSON files. The file chooser is implemented in HTML through an input field of type file. However this is not the favored display type to open a file chooser. Therefore this input field is hidden and a button is added that triggers a click event of the input field if it is selected. To prevent the import of wrong file formats, the file chooser is adjusted to only accept GeoJSON files by defining the accept=".geojson" attribute. The code to implement the file chooser can be seen in figure 46.

When the import button is clicked the import function is called. Inside this function all files that

should be opened are stored in a FileList. Subsequently each file of this FileList is imported through a FileReader and instantly converted to a JSON object. From these JSON objects the isochrones and the starting points are imported separately by the use of `var importGeojson = byteStringArray[i]["isochrones"];`, respective by `var pos = byteStringArray[i]["startpoint"];`. Finally the isochrones are added to the map by calling the addGeoJson function and the starting points by creating a marker for each of them that are shortly after each other added to the map. Via `map.setView([pos["lat"], pos["lng"]], 13);` the map is focused to the last imported starting point.

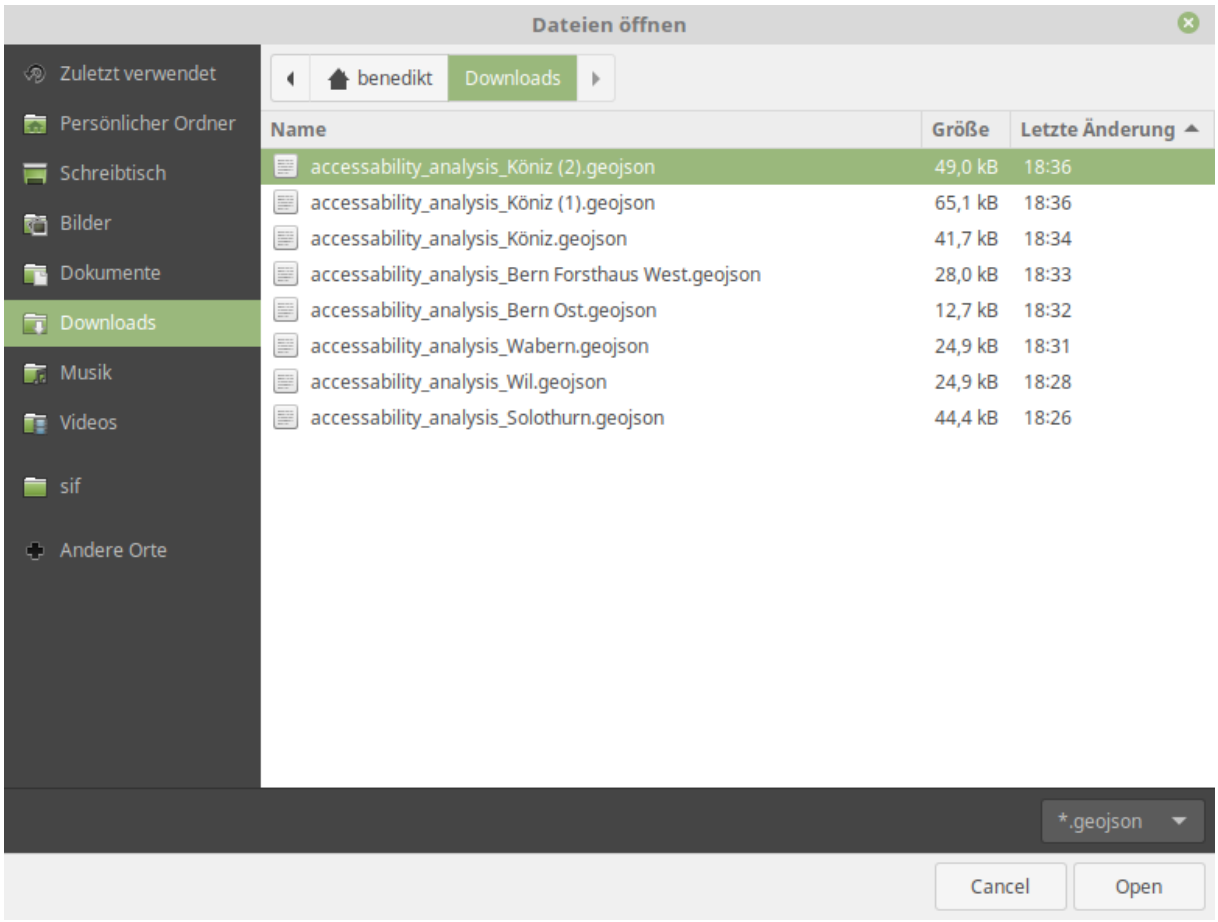

Figure 57: The file chooser displays only GeoJSON files because of the limitation accept=".geojson". For further information see chapter 7.2.1. If an already existing name is reused the new file gets an appendix like it can be seen for Köniz in this example.

```
//Import content of a GeoJSON file
function handleFileSelect(evt) {
  //check of File APIs
  checkOfFileAPI();
  //create a FileList object
  var files = evt.target.files;
    //loop through the FileList
    for (var i = 0, f; f = files[i]; i++) {
      //create a FileReader
      var reader = new FileReader();
      // Get the content of the file and parse it to JSON
        reader.onload = (function(theFile) {
            return function(e) {
              //convert dataURL to file object
                var byteString = atob(e.target.result.split(',')[1]);
              //parse byteString to JSON object
              byteStringArray = JSON.parse(byteString);
              //loop through all files
              for (i = 0; i < byteStringArray.length; i++){
                //import the isochrones
                var importGeojson = byteStringArray[i]["isochrones"];
                //call of function addGeoJson in map.js to draw the isochrones
                addGeoJson(importGeojson);
                //import the start point
                var pos = byteStringArray[i]["startpoint"];
                //check if start point is defined
                if(pos != undefined) {
                    //zoom to the start point of the accessibility analysis
                    map.setView([pos["lat"], pos["lng"]], 13);
                    //set a marker to the startpoint
                    var startmarker = L.marker([pos["lat"], pos["lng"]]);
                    //add the marker to the map
                    map.addLayer(startmarker);
                }
              }
            };
        })(f);
      // Read in the image file as a data URL.
      reader.readAsDataURL(f);
    }
}
```

Figure 58: Code of the import function. The implementation of the file chooser is visible in figure 44. At first a FileList is created to store all files that should be opened. Then the individual files are imported via a FileReader and converted to JSON objects. Isochrones are imported by `var importGeojson = byteStringArray[i]["isochrones"];` and added to the map by calling the addGeoJson function. Similar to that the starting locations are imported by `var pos = byteStringArray[i]["startpoint"];` and displayed by creating a marker for each of them that are subsequently added to the map. Moreover the map is focused to the starting point via `map.setView([pos["lat"], pos["lng"]], 13);`, in case of several starting points the map is centered to the last imported starting point, because the definition of the starting point is overwritten in each iteration of the loop.

7.4.2.3 Merge several GeoJSON files

Because the used Valhalla version cannot compute isochrones for several starting locations at the same time it is necessary to offer an opportunity to unify the results of various accessibility analyses. The result of a merge of the two fire stations in Kreuzlingen can be seen in figure 59.

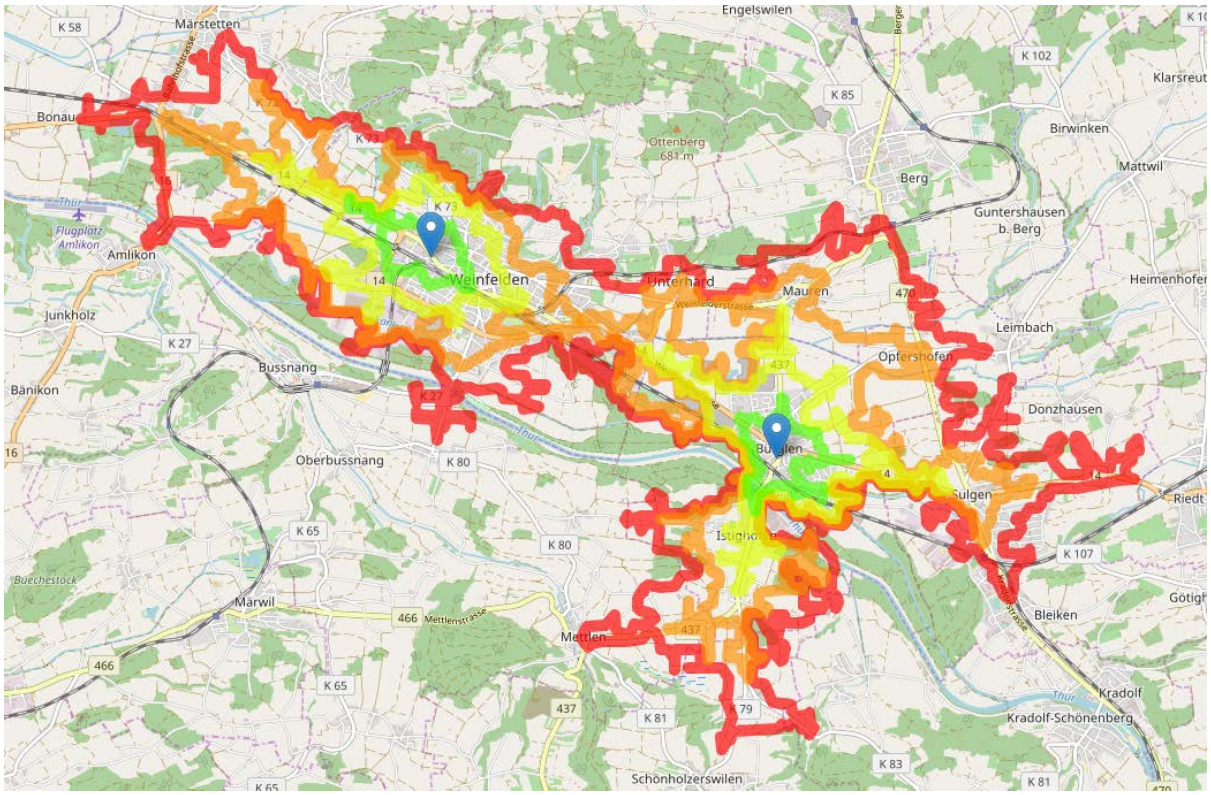

Figure 59: Merge of two fire stations in Weinfelden and Bürglen, Thurgau, Switzerland. Each fire station is represented by a marker. If the isochrones for each time level intersect each other they are merged to a single one, else two separate isochrones are drawn.

This is done with the aid of the union module of Turf.js. Therefore it is necessary to import the Turf.js library or at least the Turf.js union module. Turf.js is an open source JavaScript library. The source code of the union module, a description, and some examples can be found at GitHub (https://github.com/Turfjs/turf/tree/master/packages/turf-union). Further explanations about this module are also available at the Turf.js website (http://turfjs.org/docs/#union).

Turf.js uses the JavaScript Topology Suite (JSTS) of Björn Hartell under the hood to merge polygons.[158] Sometimes JSTS causes an error that prohibits the union of polygons. This error can be avoided by increasing the generalization value. This error is also mentioned at the GitHub repository of JSTS as follows: "In some cases you might get a TopologyException thrown as an Error. This is expected if a calculation fails due to precision issues. To resolve this issue try reducing precision in the input".[159]

As shown in figure 60 the first part of the merge function code is almost identical with the code of the import function. Just some extra variables are appended. The code below the blank line adds the isochrones to an array and converts polygons to LineStrings (Isochrones). All the markers of the starting location are attached to an additional array.

[158] MACWRIGHT, CARRIERE, BORGHI, WINSEMIUS, & DEFOSSEZ, 2017
[159] HARTELL, 2017

```
//Merge several GeoJSON files into a single one
function mergeSelected(evt) {
  //check of File APIs
  checkOfFileAPI();
  //create a FileList object
  var files = evt.target.files;
  //create the array for the unified GeoJSON
  var mergeArray = new Array();
  var markers = new Array();
  var minutesToMerge = {};
  //create an index variable and set it to zero
  var handleFiles = 0;
  // Loop through the FileList
  for (var i = 0, f; f = files[i]; i++) {
    //create a FileReader
    var reader = new FileReader();
    // Get the content of the file and parse it to JSON
    reader.onload = (function(theFile) {
      return function(e) {
        //convert dataURL to file object
        var byteString = atob(e.target.result.split(',')[1]);
        //parse byteString to JSON object
        byteStringArray = JSON.parse(byteString);

        //add the content of all geoJSON files to the array
        for (i = 0; i < byteStringArray.length; i++){
          //merge isochrones
          features = byteStringArray[i]["isochrones"]["features"];
          for(k = 0; k < features.length; k++) {
            min = features[k]["properties"]["contour"];
            var LineString;
            if (features[k]["geometry"]["type"] == "Polygon"){
                //convert polygons to LineStrings
                LineString = turf.polygonToLineString(features[k]["geometry"]);
                features[k]= LineString;
              }
            if (minutesToMerge[min] == undefined) {
              minutesToMerge[min] = new Array();
            }
            minutesToMerge[min].push(features[k]["geometry"]["coordinates"]);
          }
          //merge markers
          var startmarker = L.marker([byteStringArray[i].startpoint["lat"],
                                      byteStringArray[i].startpoint["lng"]]);
          markers.push(startmarker);
        }
```

Figure 60: First part of the merge function. Except of adding a few variables the code above the blank line is equal to the code used in the import function (see figure 58). Later on the content of all GeoJSON files is stored in a single array called mergeArray. In doing so polygons are converted to LineStrings.

Subsequently the variable handleFiles is counted one up each time the loop is traversed. This is needed to check if all files are added to the array. If this is the case the isochrones of the identical time interval are merged in the union variable and then added to the completeMerged array. Figure 61 shows the just explained code. Because the union module of Turf.js uses polygons the input to this and the output of this have to be converted from LineString to polygon and from polygon to LineString. However this is tricky, because if the polygons do not intersect each other they are stored as Features in an additional FeatureCollection within the usually FeatureCollection, however this contradicts the GeoJSON format. Therefore they are not visualized. This problem is solved by eliminating these redundant FeatureCollections as can be seen in figure 64.

```
//count handleFiles one up each time the for loop is traversed
handleFiles++;
//Create an export of the merged GeoJSON file if all GeoJSON files are added
//check if all files are added to the merge
if (handleFiles == files.length) {

  var completeMerged = {"features":[]};
  $.each(minutesToMerge, function(key_min, value_array) {
    //create a variable for the unified isochrones
    var union = undefined;
    //loop through all selected files that should be merged
    for (fileNr = 0; fileNr < value_array.length; fileNr++) {
      tmp = turf.polygon([value_array[fileNr]]);
      //check if the variable union is undefined
      if (union == undefined) {
        //set the variable union to value of the first input file
        union = tmp;
        continue;
      }
      //merge isochrones via Turf.js union
      unionTmp = turf.union(union, tmp);
      union = unionTmp;
    }
    //check if union is not undefined
    if(union != undefined) {
      //convert polygons to LineStrings
      var line = turf.polygonToLineString(union);
      union.properties = {
        "color":"rgb(255,0,0)",
        "contour":key_min,
        "opacity":0.33,
      };
      completeMerged["features"].push(union);
    }
  });
  //switch the order of the features
  completeMerged["features"].reverse();
```

Figure 61: Resumption of the code of the merge function. In this code extract the variable handleFiles is counted one up each time the loop is traversed to be able to check when all files are added to the array. Then successively the isochrones are merged in the union variable separate for each minute. Because the output of Turf.js union are polygons, these polygons are converted to LineStrings.

Like figure 62 shows the merged isochrones are then colored in the same manner as already described earlier. After that the isochrones and the markers are added to the map via the addGeoJson function. Finally the result of the merge is exported and saved in a file called accessibility_analysis_merged.geojson. This functionality can be seen in figure 63.

```
for(i=1; i < completeMerged["features"].length; i++) {
  var color = HSVtoRGB((96/255)*
  (completeMerged["features"].length-completeMerged["features"][i].properties.contour)/(completeMerged["features"].length), 1, 1);
  completeMerged.features[i].properties.color = "#" +
  toPaddedHexString(color.r, 2)+toPaddedHexString(color.g, 2)+toPaddedHexString(color.b, 2);
}
addGeoJson(completeMerged);
for(i=0;i<markers.length;i++) {
  map.addLayer(markers[i]);
}
```

Figure 62: In this extract of the merge function the resulting isochrones are colored from green to red depending on the duration like described in chapter 7.2.2.2. The merged isochrones are visualized at the map by calling the addGeoJSON function. The markers for the start locations are merged by simply aggregate in an array.

```
            //stringify the merged GeoJSON to be able to save it into a GeoJSON file
            exportFile = [{"isochrones":completeMerged}];
            for(i=0;i<markers.length;i++) {
              exportFile.push({"startpoint":markers[i].getLatLng(),"isochrones":{"features":[]}});
            }
            var mergeCompleted = JSON.stringify(exportFile);
            //Create export
            //create a new <a> element
            var link = document.createElement('a');
            //define the file name of export file (accessability_analysis_merged.geojson)
            link.download = 'accessability_analysis_merged.geojson';
            //define the uri
            link.href = 'data: text/json;charset=utf-8,' + encodeURIComponent(mergeCompleted);
            //Firefox requires the link to be in the body
            document.body.appendChild(link);
            //simulate click
            link.click();
            //remove the link when done
            document.body.removeChild(link);
          }
        };
      })(f);
      // Read in the image file as a data URL.
      reader.readAsDataURL(f);
    }
}
```

Figure 63: This extract of the merge function shows the export of the merged files as a file called accessibly_analysis_merged.geojson. Therefore the result of the merge is converted to a JSON string and the headings "isochrones" and "startpoint" are added. This is similar to the already described procedure in the export function.

```
//add isochrones to map
function addGeoJson(json) {
  if (json.features){
      //check if the last element (always element for contour 1) has the type FeatureCollection
      if (json.features[json.features.length - 1].type == "FeatureCollection"){
          //create a new object
         converted = { "features" : [], "type" : "FeatureCollection" };
         //loop through json
         json.features.forEach(function(x) {
           //check if current element has the type FeatureCollection
           if (x.type == "FeatureCollection"){
             //loop through x
             x.features.forEach(function(y) {
               //copy the properties of the feature Collection to the single features
               y.properties = x.properties;
               //add the current element to the object
               converted.features.push(y);
             })
           }
           //check if the current element has the type Feature
           else if(x.type == "Feature"){
             //create a new object
             y={};
             //copy the properties to the single features
             y.properties = x.properties;
             //copy the type to the single features
             y.type = x.type;
             //copy the geometry to the single features
             y.geometry = x.geometry;
             //add the current element to the object
             converted.features.push(y);
           }
         });
         //set json to converted
         json = converted;
         console.log("converted:");
         console.log(json);
      } else{
         console.log("nothing to convert")
      }
  }
  else
  {
    console.log("wrong GeoJSON type error at map.js function addGeoJson");
  }
```

Figure 64: In this extract of the addGeoJson function potentially existing redundant FeatureCollections are eliminated. For that purpose at first it is checked whether the last element of the "json" array (this contains all the isochrones) is a FeatureCollection. The last element is always the 1 minute isochrone. If this is the case a new object called "converted" is created in which the Features should be stored later on. Then all elements of the json array are investigated. If an element is a FeatureCollection the in it included Features receive the properties of the FeatureCollection and are stored in "converted". Else a new object "y" is created with the properties of the Feature and attached to "converted". Finally "json" is overwritten by "converted".

Merging the results of several accessibility analysis disable the determination of the travel time from a certain starting location.

7.4.3 Inspection of the result

To evaluate the result of the accessibility analysis it is necessary to compare the reachable area with the boundary of the area that should be reached. Thereby it is possible to visually check whether all required areas are accessible in the prescribed time interval. Figure 65 shows how to add the city boundaries for a city. Then a comparison between the isochrones and the city boundary is possible as it is visible in figure 66.

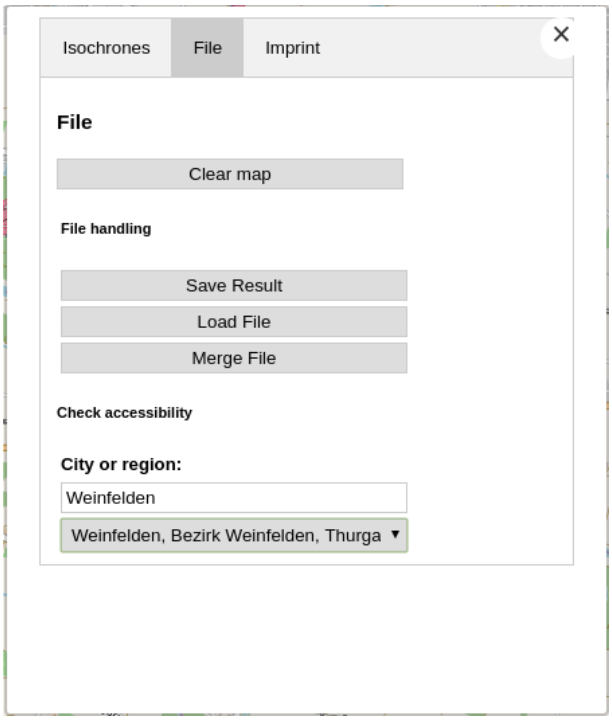

Figure 65: The name of the city has to be inserted in the text field and then the desired entry can be selected in the drop-down menu to display the city boundaries.

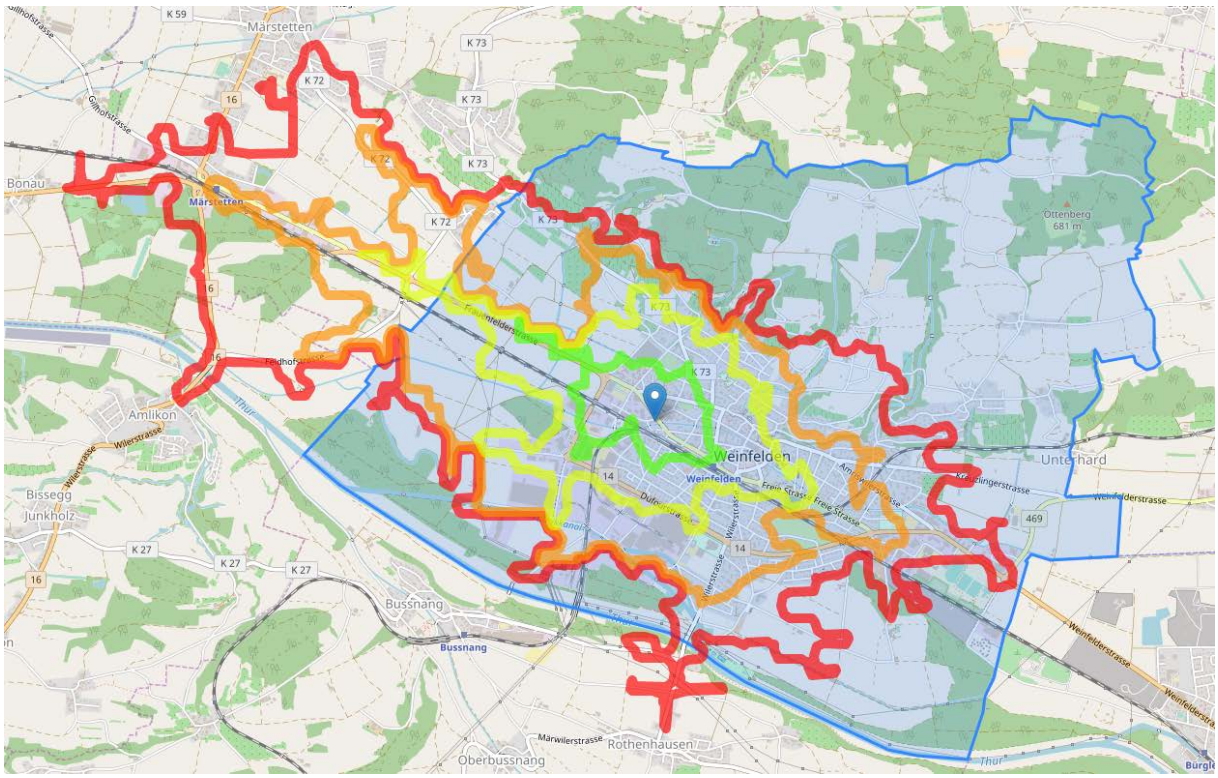

Figure 66: Four isochrones around the fire station of Weinfelden, Thurgau, Switzerland. To check if all necessary areas are reachable the boundary of Weinfelden is added as a blue polygon. As a visually comparison shows almost all built-up areas of Weinfelden are within the four minute isochrone, but there are also large areas of Weinfelden outside the four minute isochrone.

There are two different possibilities to add these boundaries to the map. On the one hand Overpass can be used, which adds all borders to the map for the screen extent and on the other hand it is possible to search for a special boundary via Nominatim and only display this single border.

Although the Overpass way is easier to implement it has the disadvantage that all boundaries in the screen extent are visualized identically. Therefore it is not possible to distinguish between the borders of different cities, which is for instance important if a city consists of several disconnected areas.
For this reason the Nominatim way is used. The user inputs the name of the city or region for which they want to see the boundary and then they can choose from the drop-down list below the input field. This is necessary because it is possible that there are different objects that have the same name in OSM. The input field and the drop-down menu for adding the boundaries of a certain city or area are visible at the bottom of figure 42.

In the function getBoundaryURL the URL to request the boundaries from Nominatim is built by attaching the name the user entered to the fixed part of the URL. If there are many objects with this name in OSM, Nominatim returns up to 10 results by default. This number can be increased or decreased by defining a number for the optional limit attribute. At the end of this function the inspect function is called to send the request to Nominatim as well as grab the names of the several objects and add them to the drop-down menu. Figure 67 shows the function getBoundaryURL and figure 68 displays the function inspect. The getBoundaryURL function is called when the content of the name input field (id = city_value) has changed.
Before the boundary of the selected option is added to the map, potentially existing boundaries of previous requests have to be deleted to avoid confusions. The code for these functions can be seen in figure 69. These functions are called when the user selects are an option of the drop-down menu.

```
//send a request to Nominatim to get the boundaries of a specified city or region
function getBoundaryURL(){
  //build url
  //first, fixed part of the url
  boundaryURL = 'http://nominatim.openstreetmap.org/search?format=jsonv2&polygon_geojson=1&q=';
  //add the wanted term to the first, fixed part
  boundaryURL += city;
  //call inspect function
  inspect();
}
```

Figure 67: Code for building the URL to request the boundary of a city or region from Nominatim. Therefore the name that the user entered is attached to a fixed part of the URL. Finally the inspect function is called.

```
//send a request to Nominatim (boundaryURL) and display the names (display_name) of the response in the drop-down menu
function inspect(evt) {
  // send a HTTP GET request to Nominatim by using jQuery
  $.get(
    boundaryURL,
    function(data) {
      //copy content of data into a global variable to be able to use the content outside of this function
      copyData = data;
      //get the selected city out of the available possible cities in the drop-down menu
      select = document.getElementById("selectedCity_value");
      //set the first entry of the select drop-down menu to "choose correct city"
      select.innerHTML = "<option value=100>choose correct city</option>";
      //insert all display_name names into the drop-down menu to enable the user to select the right city
      for(i = 0; i < data.length; i++) {
        //display only the name (display_name) of the different answers of the response
        var options = data[i]["display_name"];
        //create an option listElement
        var listElement = document.createElement("option");
        //set the textContent of this listElement to the corresponding display_name entry of the response
        listElement.textContent = options;
        //set the value of this listElement to i (number of loops up to this)
        listElement.value = i;
        //add this entry to the drop-down list
        select.appendChild(listElement);
      }
    }
  );
}
```

Figure 68: Code to send the in the getBoundaryURL built request to Nominatim. The response is stored in the variable data. To grab the names of the objects the line `var options = data[i]["display_name"];` is used. This name is subsequently used to allow the user to choose the correct option in the drop-down menu.

```
//draw the chosen city onto the map
function drawCityBoundary(){
  //avoid error if user clicks at select correct city entry of drop-down menu
  if (selectedCity == 100) {
    alert("no city is chosen");
  }else{
    //add boundary coordinates to the variable boundaryLayer
    boundaryLayer = new L.GeoJSON(copyData[selectedCity].geojson);
    //add boundaryLayer to the map
    map.addLayer(boundaryLayer);
  }
}

//delete boundary layer from the map
function deleteBoundaryLayer(){
   //check if boundaryLayer exists
  if(boundaryLayer != null){
    //delete boundaryLayer
    map.removeLayer(boundaryLayer);
  }
}
```

Figure 69: Code of the functions to add the boundaries to the map and to delete the boundaryLayer.

7.4.4 Geocoder

The geocoding functionality is embedded only as a service of Nominatim geocoder, because a local implementation of Nominatim would by far exceed the storage and RAM capacity of the used machine. As mentioned in the Nominatim installation guide about 500 GB of hard disk space and at least 32 GB RAM is necessary to install Nominatim for geocoding worldwide. This would take more

than 2 days to import the required data if a SSD is used, otherwise up to 7-8 days are needed.[160] The source code and an installation guide of Nominatim can be found at https://github.com/openstreetmap/Nominatim.

In this project the geocoding functionality is part of the basic functions and represented by the button with the magnifier icon on it as shown in figure 38. Figure 40 shows the geocoding menu if this button is selected. It consists of a search box and button to start the query. The source code of the geocoder is contained in the Control.OSMGeocoder.js and Control.OSMGeocoder.css files. Moreover the geocoding function is added to the map in bodyLoadScript.js as Leaflet control element.

7.4.5 Clear map function

The user should have the possibility to clear the map content without refreshing the page by pressing the F5 key on the keyboard or activate the refresh function of the browser, because some browsers store some information that have been inserted to a website in their cache and therefore do not delete this. This is solved by the clear map function that is called by selecting the Clear map button in the File tab of the settings menu. All map layers are deleted (`map.removeLayer(layer)`) as well as the text input fields and some variables are exhausted or reset to undefined (for example `document.getElementById("giveId").value=""` and `startLocation="null"`). A special case is the last variable shown in figure 70 because it is an array.

```
//clear map
function clearAllFromMap(){
  //clear all layers
  map.eachLayer(function (layer) {
    if(layer != osmLayer) {
      map.removeLayer(layer);
    }
  });

  //clear text fields
  document.getElementById("giveId").value = "";
  document.getElementById("city_value").value = "";

  //clear variables
  copyIsochrones= "";
  copyPosition= "";
  isochrones="";
  position="";
  startLocation=null;
  select.innerHTML = "<option value=100>choose correct city</option>";
}
```

Figure 70: The code of the clear map function is located in the file map.js.

[160] HOFFMANN & TOBIAS, 2017

8 Testing/ verification

To ensure the correct display of the frontend on the most frequently used web browsers as well as different operating systems, the frontend has been tested on various browsers on different operating systems. Firefox and Chrome were tested on Windows as well as on Linux and Internet Explorer was tested on Windows. No significant differences are visible between all tested browsers.

Several test persons had no problems to use the website. The website is clearly structured and intuitive to handle. It is possible to compute isochrones by only defining the starting location. Just a few clicks are necessary to adapt the calculation by changing the default settings or to export the result of the accessibility analysis. The union of GeoJSON files with different settings is possible. The inspection of the reachable area with the help of city boundaries is useful. The presentation of the isochrones in a color gradient from green to red shows at the first glance which areas are easy to reach and which are difficult to reach in a predefined time interval.

All targeted functions are implemented and are well-working except of the custom profiles. Therefore the default profiles auto and truck of Valhalla are used.

The computed result of Valhalla is reviewed by comparing the travel time with the duration of a route between the starting location and a couple of points at the 20 minutes isochrone. For comparison the routes are calculated via GoogleMaps, GraphHopper and Mapzen Turn-by-Turn. No extraordinary large discrepancies are detected.

But there are also some problems. Because of the raster approach of Valhalla the isochrones do not match the road network all the time as can be seen in figure 71.

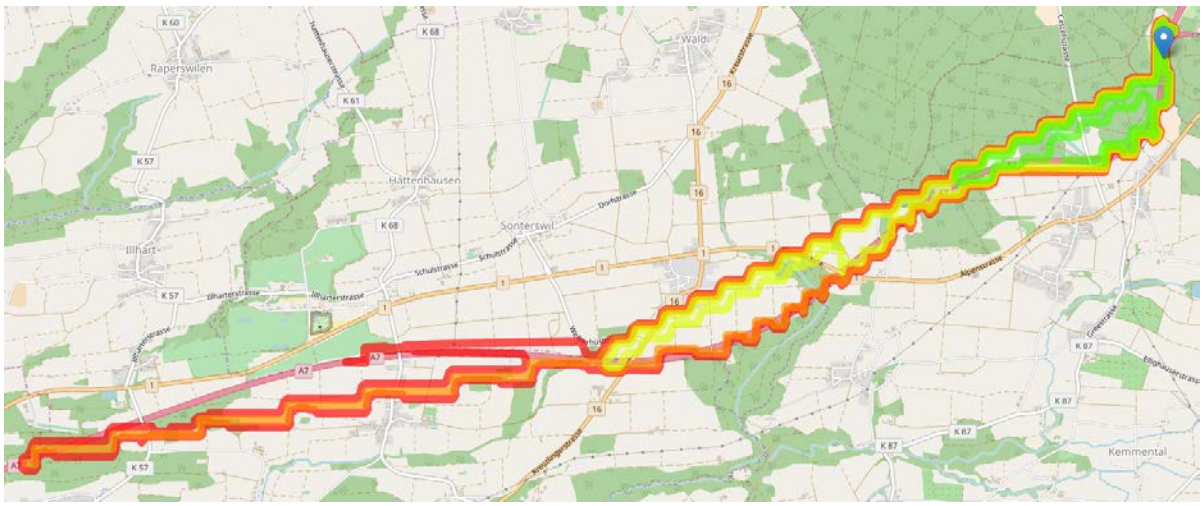

Figure 71: The discrepancies between the isochrones and the road network can clearly be seen at the motorway A7.

Valhalla has also problems to compute isochrones in case there are tunnels. Some examples of this problem can be seen in figure 72 and figure 73.

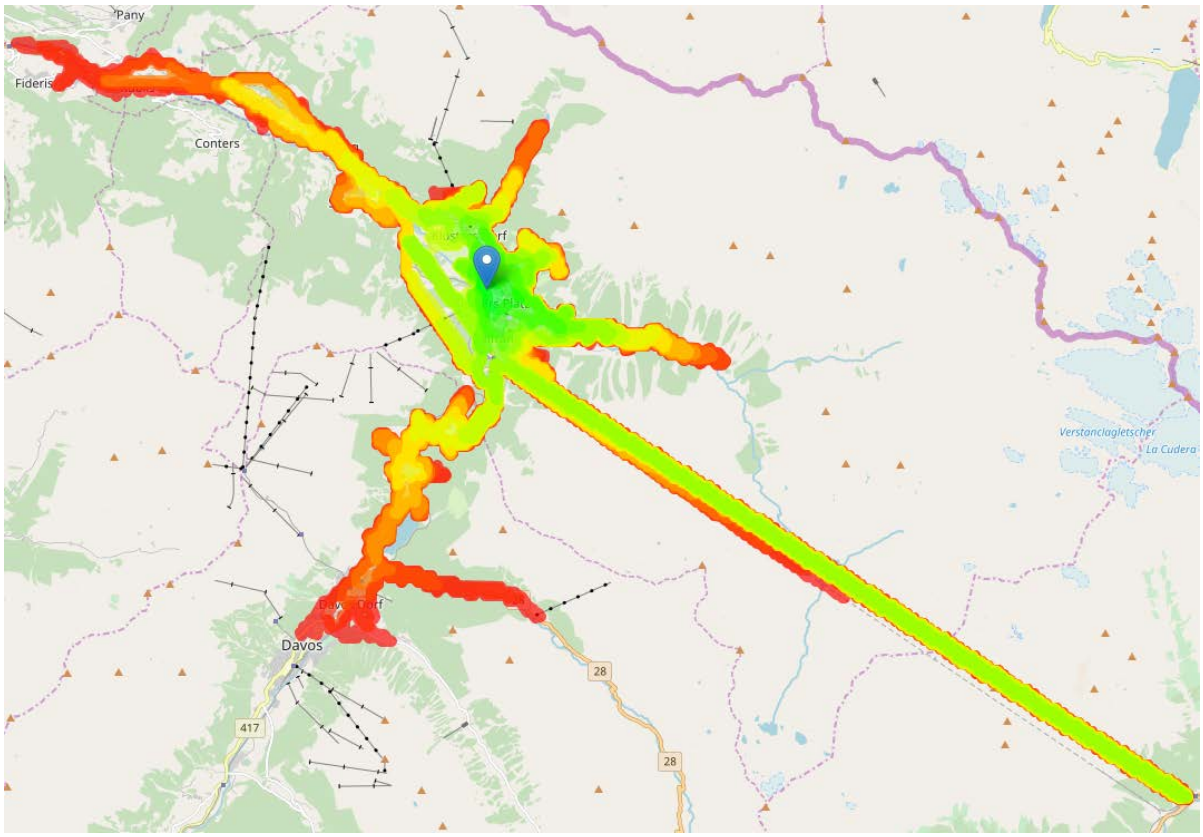

Figure 72: Isochrone computing problem of Valhalla at the calculation of the reachable area for the fire station Klosters Platz, Grisons, Switzerland. The complete tunnel at the right side of the image is marked as reachable, but it is obvious this cannot be the case.

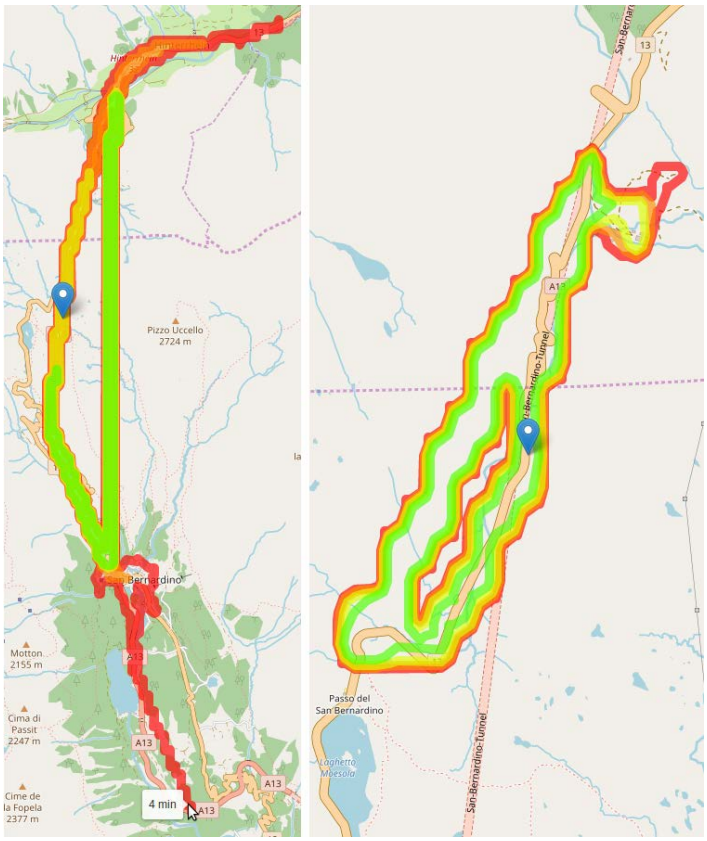

Figure 73: Wrong results in case the starting point is close to a tunnel and in a tunnel.

9 Summary and outlook

9.1 Summary

In this project the task was to develop a website to perform accessibility analyses via open source routing algorithms and based on open geodata. This aim has been achieved.

Four different open source routing algorithms were tested and the suitability of OpenStreetMap data was investigated.
Although the data of OpenStreetMap is not complete it is suitable for performing accessibility analyses. Generally the quality of OpenStreetMap data is not homogenous. It depends on the country as well as whether the area is urban or rural. In some western cities it is already better than other map providers like Google Maps, in some countries it is the best available geodata despite of the incompleteness.
The routing algorithms GraphHopper, openroutservice.org, OSRM-isochrone and Valhalla were investigated. These four routing algorithms only OSRM-isochrone and Valhalla offer an open source isochrone computation and are usable without limitations. Because OSRM-isochrone is not maintained any more, Valhalla has been selected as a backend.
To allow the user a simple interaction with the Valhalla backend to perform accessibility analyses a website was programmed as a frontend. This website enables the user to adapt the calculation to their needs. Among others they can choose between different routing profiles as well as define a maximum travel time. The travel time is part of the response time and varies between the different emergency services as well as for different types of emergencies. In addition to that this configuration option enables the user to compute the alarm response area of special vehicles. The result of an accessibility analysis can be exported to a GeoJSON file. It is also possible to import a GeoJSON file created earlier and display it on the OpenStreetMap map. The execution of an accessibility analysis with several starting points is not possible because on the one hand the used Valhalla version does not support it and on the other hand it is not wanted, because an accessibility analysis with several starting points cannot be performed if different routing profiles or travel times are requested. Therefore a merge function is implemented instead to unify the results of several accessibility analyses into a single result file. A validation of the result with boundaries of counties, cities, and boroughs is possible. Hereby it is easy to recognize which areas are reachable within the predefined time interval and which not. For this purpose Nominatim is used. Although Nominatim is open source and can be used self-hosted it is used as service in this project, because of the lack of storage capacity.
Translations of the used discipline-specific words are added as a glossary in the appendix to facilitate the understanding of the report.

9.2 Outlook and possible improvements

Even though the use of OpenStreetMap data already yields good results, the completeness and the quality will increase in the future.
A new evaluation of the routing algorithms could also be reasonable in the future, because most of the algorithms are under active development. For example the newest version of Valhalla offers accessibility analyses for several starting points and new algorithms like Mapbox-isochrone are published.

To offer this project to people worldwide it has to run on a server instead of just locally on a single machine. Therefore a server is needed as well as some little modifications. For example the URL address has to be changed to the address of the server and a short function has to be written in Bash that restarts the server if it crashes.

Custom profiles have to be added for the classes defined in chapter 6.3 and explained in chapter 7.3.

Although the result of the accessibility analyses could be more accurate if real time traffic information or at least average time of day dependent traffic conditions are taken into account, but the accessibility analysis should provide a general overview instead of a snapshot. Therefore this could only be an additional analysis.
Also the implementation of additional functions to the frontend is conceivable. For example the centering of the map to the selected boundary is possible, however this could also be disadvantageous if not the desired boundary is chosen and therefore the map is centered far off the desired city. The possibility to save the result in other data formats than GeoJSON, for instance the Shape format which is widely used in Geographic Information Systems (GIS), would also be an advantage. Moreover a multilingual website will facilitate the use for non English speaking people. The backend can also be improved by considering the geometry of the road geometry during the travel time computation, because on curvy roads vehicles cannot drive as fast as on straight roads. Partially this is already implemented in Valhalla by reducing the velocity near crossings.

List of literature

AMELUNXEN, C. (2010). *An Approach to gecoding based on volunteered Spatial Data .* Heidelberg: University of Heidelberg.

ARBEITSGEMEINSCHAFT DER LEITER DER BERUFSFEUERWEHREN. (2015). *Qualitätskriterien für die Bedarfsplanung von Feuerwehren in Städten.* Bonn: Arbeitsgemeinschaft der Leiter der Berufsfeuerwehren(AGBF).

ARCGIS ONLINE HELP. (2017). *Create Drive-Time Areas*. Retrieved August 21, 2017, from https://doc.arcgis.com: https://doc.arcgis.com/en/arcgis-online/analyze/create-drive-time-areas.htm

BARRINGTON-LEIGH, C., & MILLARD-BALL, A. (2017, August 12). *The world's user-generated road map is more.* Retrieved August 21, 2017, from http://journals.plos.org: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0180698

BARTELME, N. (2005). *Geoinformatik; Modelle, Strukturen, Funktionen.* Berlin, Heidelberg: Springer Verlag.

BAYRISCHES ROTES KREUZ. (2017). *Produktentwicklung und Qualität - Technik im Rettungsdienst*. Retrieved July 28, 2017, from https://rettungsdienst.brk.de: https://rettungsdienst.brk.de/technik

BILL, R. (2010). *Grundlagen der Geo-Informationssysteme* (5th edition ed.). Berlin and Offenbach: Herbert Wichmann Verlag.

BOURKE, P. (1987, July). *CONREC - A Contouring Subroutine*. Retrieved August 11, 2017, from http://paulbourke.net: http://paulbourke.net/papers/conrec/

BOURKE, P. D. (1987, June). A Contouring Subroutine. *BYTE - The small systems journal* , pp. 143-150.

BROVELLI, M. A., MINGHINI, M., MOLINARI, M. E., & ZAMBONI, G. (2016). POSITIONAL ACCURACY ASSESSMENT OF THE OPENSTREETMAP BUILDINGS LAYER THROUGH AUTOMATIC HOMOLOGOUS PAIRS DETECTION: THE METHOD AND A CASE STUDY. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* , pp. 615-620.

BUNDESANSTALT TECHNISCHES HILFSWERK. (2017). *Fahrzeuge.* Retrieved July 28, 2017, from www.thw.de: https://www.thw.de/DE/Einheiten-Technik/Ausstattung/Fahrzeuge/fahrzeuge_node.html

BUTLER, H., DALY, M., DOYLE, A., GILLIES, S., HAGEN, S., & SCHAUB, T. (2016, August). *The GeoJSON Format.* Retrieved July 17, 2017, from www.tools.ietf.org: https://tools.ietf.org/html/rfc7946

BUTZER, A. S. (2017, June 19). *OpenRouteService docs*. Retrieved August 17, 2017, from https://github.com/GIScience/openrouteservice-docs: https://github.com/GIScience/openrouteservice-docs

CHISTOPOULOU, K., ELLUL, C., & FRAM, C. (2015). *Assessing the quality of OpenStreetMap building data and searching for a proxy variable to estimate OSM building data completeness.* London: Dept. of Civil, Environmental and Geomatic Engineering, University College London.

DARA-ABRAMS, D., DILUCA, G. D., KIRSTEN, KREISER, K., MEHYAR, M., NESBITT, D., et al. (2017, August 10). *valhalla_build_config*. Retrieved August 29, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla/blob/master/scripts/valhalla_build_config

DESTATIS. (2016, September 30). *Gemeindeverzeichnis*. Retrieved March 31, 2017, from www.destatis.de: https://www.destatis.de/DE/ZahlenFakten/LaenderRegionen/Regionales/Gemeindeverzeichnis/Administrativ/Archiv/GVAuszugQ/AuszugGV3QAktuell.xls?__blob=publicationFile

DEUTSCHES BUNDESAMT FÜR BEVÖLKERUNGSSCHUTZ UND KATASTROPHENHILFE. (2005). *www.bbk.bund.de/DE/Servicefunktionen/Glossar/_function/glossar.html*. Retrieved March 21, 2017, from BBK - Bundesamt für Bevölkerungsschutz und Katastrophenhilfe: http://www.bbk.bund.de/DE/Servicefunktionen/Glossar/_function/glossar.html?lv3=1948880&lv2=4968152

DEUTSCHES BUNDESMINISTERIUM DES INNERN. (2017). *http://www.bevoelkerungsschutz-portal.de/BVS/DE/Zustaendigkeiten/Feuerwehr/feuerwehr_node.html*. Retrieved March 21, 2017, from Bevölkerungsschutz-Portal des Bundesministeriums des Innern: http://www.bevoelkerungsschutz-portal.de/BVS/DE/Zustaendigkeiten/Feuerwehr/feuerwehr_node.html

DIBBELT, J., STRASSER, B., & WAGNER, D. (2015). *Customizable Contraction Hierarchies* . Karlsruhe: Karlsruhe Institut of Technology.

DIJKSTRA, E. W. (1959, June 11). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematlk* , pp. 269-271.

DIJKSTRA, E. W. (2001, August 2). An Interview With Edsger W. Dijkstra. (P. L. Frana, Interviewer)

DILUCA, K., GEARHART, D., KNISELY, G., KREISER, K., & NESBITT, D. (2015, June 04). *Introducing Valhalla*. Retrieved August 04, 2017, from https://mapzen.com: https://mapzen.com/blog/introducing-valhalla/

DILUCA, K., GEARHART, D., KNISELY, G., KREISER, K., NESBITT, D., & WILTON, A. (2017, August 22). *Release notes*. Retrieved August 29, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla-docs/blob/master/release-notes.md

DIN DEUTSCHES INSTITUT FÜR NORMUNG E.V. (2017). *DIN Standards*. Retrieved July 27, 2017, from http://www.din.de: http://www.din.de/en/about-standards/din-standards

DIN-NORMENAUSSCHUSS FEUERWEHRWESEN. (2016, November 10). *Feuerwehrfahrzeugkonzeption des DIN-FNFW - Feuerwehrfahrzeug-Typenliste der gängigsten genormten Fahrzeuge.* Retrieved July 27, 2017, from www.din.de: http://www.din.de/blob/118608/dff262dbff60605342f62268416b13e6/feuerwehrfahrzeug-typenliste-21-fassung-2016-11-data.pdf

DOMSCHKE, D. W. (1981). *Logistik: Transport. Grundlagen, lineare Transport- und Umladeprobleme.* München: R. Oldenbourg Verlag GmbH.

DOUGLAS, D. H., & PEUCKER, T. K. (1973, December). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian cartographer* , pp. 112-122.

DUDEN. (2017). *Netzwerk, das*. Retrieved July 18, 2017, from www.duden.de: http://www.duden.de/rechtschreibung/Netzwerk

ELLERSIEK, T. (2017, April 29). *OpenRouteService with new API, functions and look*. Retrieved August 17, 2017, from http://k1z.blog.uni-heidelberg.de: http://k1z.blog.uni-heidelberg.de/2017/04/29/openrouteservice-with-new-api-functions-and-look/

ESRI. (2017). *Which areas are within four minutes of a fire station?* Retrieved August 21, 2017, from http://desktop.arcgis.com: http://desktop.arcgis.com/en/analytics/case-studies/which-areas-are-within-four-minutes-of-a-fire-station.htm?lg=en

EUROPEAN COMITTEE FOR STANDARDIZATION: CEN AFFILIATES. (2017). *CEN Affiliates*. Retrieved July 26, 2017, from https://standards.cen.eu: https://standards.cen.eu/dyn/www/f?p=CENWEB:9:::NO:::

EUROPEAN COMITTEE FOR STANDARDIZATION: CEN MEMBERS. (2017). *CEN Members*. Retrieved July 26, 2017, from https://standards.cen.eu: https://standards.cen.eu/dyn/www/f?p=CENWEB:5:::NO:::

EUROPEAN COMITTEE FOR STANDARDIZATION: LIST OF COMPANIONS. (2017). *List of Companion Standardization Bodies*. Retrieved July 26, 2017, from https://standards.cen.eu: https://standards.cen.eu/dyn/www/f?p=CENWEB:60:::NO:::

FISCHER, C. (2013, August). *www.christian-fischer.info/2013/08*. Retrieved March 21, 2017, from www.christian-fischer.info: http://www.christian-fischer.info/2013/08/der-selbstbetrug-mit-der-hilfsfrist/

FISCHER, M. M. (2003). *GIS AND NETWORK ANALYSIS.* ERSA conference papers, European Regional Science Association.

FISCHER, T. (2017, July 22). (B. Futterer, Interviewer)

FOSTER, C., GEARHART, D., GLENNON, A., HARDS, B., KNISELY, G., KREISER, K., et al. (2017, July 31). *Valhalla*. Retrieved August 18, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla

FREIWILLIGE FEUERWEHR FINNENTROP. (2010). *Erreichbarkeitsanalyse*. Retrieved March 30, 2017, from https://www.feuerwehr-finnentrop.org: https://www.feuerwehr-finnentrop.org/wissenswertes/erreichbarkeitsanalyse

FUNKE, S. (2017). *Contraction Hierarchies briefly explained.* Stuttgart.

GEISBERGER, R. (2008). *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.* Karlsruhe: Institut für Theoretische Informatik, Universität Karlsruhe (TH).

GEOFABRIK. (2016). *Map Compare*. Retrieved March 30, 2017, from http://tools.geofabrik.de: http://tools.geofabrik.de

GEOFABRIK. (2016). *OpenStreetMap Data Extracts.* Retrieved August 29, 2017, from http://www.geofabrik.de: http://download.geofabrik.de

GEOJSON WORKING GROUP. (n.d.). *GEOJSON.* Retrieved August 07, 2017, from http://geojson.org: http://geojson.org/

GORKI, H. F., & PAPE, H. (1987). *Stadtkartographie* (Vol. III/1). (E. Arnberger, Ed.) Wien: Deuticke, Franz.

GÖTZ, M., & ZIPF, A. (2012). *OpenStreetMap in 3D – Detailed Insights on the Current Situation in Germany* . Avignon: AGILE 2012.

GOULD, P. R. (1969). *Spatial diffusion.* Washington D.C.: Association of American Geographers, Comission on College Geography.

GRAPHHOPPER. (A). *About us.* Retrieved August 16, 2017, from www.graphhopper.com: https://www.graphhopper.com/about-us/

GRAPHHOPPER. (B). *GraphHopper Directions API Live Examples*. Retrieved August 16, 2017, from https://graphhopper.com: https://graphhopper.com/api/1/examples/#isochrone

GRAPHHOPPER. (C). *Pricing GraphHopper Directions API*. Retrieved August 16, 2017, from www.graphhopper.com: https://www.graphhopper.com/pricing/

HAKE, G., GRÜNREICH, D., & MENG, L. (2002). *Kartographie - Visualisierung raum-zeitlicher Informationen* (8. vollständig neu bearbeitete und erweiterte Auflage ed.). Berlin/ New York: de Gruyter Lehrbuch.

HAKLAY, M., BASIOUKA, S., ANTONIOU, V., & ATHER, A. (2010, November). How Many Volunteers Does it Take to Map an Area Well? *The Cartographic Journal* , pp. 315–322.

HÄRDI, R. (2017, March 2017). *2017 - Feuerwehreinsatzkarten mit OSM*. Retrieved August 27, 2017, from www.youtube.com: https://www.youtube.com/watch?v=1__IjaP1EY8

HART, P. E., NILSSON, N. J., & RAPHAEL, B. (1968, July 2). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics* , pp. 100-107.

HARTELL, B. (2017, July 8). *bjornharrtell/jsts*. Retrieved September 5, 2017, from https://github.com/bjornharrtell: https://github.com/bjornharrtell/jsts

HECHT, R., KUNZE, C., & HAHMANN, S. (2013, November 11). Measuring Completeness of Building Footprints in OpenStreetMap over Space and Time. *International Journal of Geo-Information* , pp. 1066-1091.

HECHT, R., KUNZE, C., & HAHMANN, S. (2014, March 6). *Zur Vollständigkeit des Gebäudedatenbestandes von OpenStreetMap.* Retrieved August 22, 2017, from www.dgfk-mittelrhein.de: http://www.dgfk-mittelrhein.de/files/f-vortrag06032014.pdf

HERLOCKER, M., GIRARD, F., NIKLAUS, P., VAN DOORN, F., & RODRIGO, F. (2016, October 28). *osrm-isochrone*. Retrieved August 17, 2017, from https://github.com/mapbox/osrm-isochrone: https://github.com/mapbox/osrm-isochrone

HOFFMANN, S., & TOBIAS, M. (2017, June 25). *Nominatim installation*. Retrieved August 30, 2017, from https://github.com/openstreetmap/Nominatim: https://github.com/openstreetmap/Nominatim/blob/master/docs/Installation.md

HURNI, L., IOSIFESCU, I., & MÜLLER, F. (2015). *Assessment and Visualization of OSM Building Footprint Quality.* Rio de Janeiro: 27th International Cartographic Conference.

HÜSCH, F. (2017, July 24). (B. Futterer, Interviewer)

ITO MAP. (2017, August 25). *Speed limits*. Retrieved August 25, 2017, from http://product.itoworld.com: http://product.itoworld.com/map/124?lon=64.61833&lat=16.77172&zoom=4&open_sidebar=map_key&fullscreen=true

KARICH, P. (2017, August 14). *Flexible routing at least 15 times faster*. Retrieved August 16, 2017, from www.graphhopper.com: https://www.graphhopper.com/blog/2017/08/14/flexible-routing-15-times-faster/

KARICH, P. (2013a, November 26). *GraphHopper an Open Source and Flexible Alternative to Google Maps API*. Retrieved August 16, 2017, from https://karussell.wordpress.com/: https://karussell.wordpress.com/2013/11/26/graphhopper-an-open-source-and-cost-efficient-alternative-to-google-maps-api/

KARICH, P. (2013, July 22). *GraphHopper Maps 0.1 – High Performance and Customizable Routing in Java*. Retrieved August 16, 2017, from www.graphhopper.com: https://www.graphhopper.com/blog/2013/07/22/graphhopper-maps-high-performance-and-customizable-routing-in-java/)

KARICH, P. (2017, May 31). *GraphHopper Routing Engine 0.9 Released*. Retrieved August 16, 2017, from www.graphhopper.com: https://www.graphhopper.com/blog/2017/05/31/graphhopper-routing-engine-0-9-released/

KARICH, P. (2017, July 27). *Isochrone API*. Retrieved August 16, 2017, from https://github.com/graphhopper: https://github.com/graphhopper/directions-api/blob/master/isochrone.md

KARICH, P. (2017, July 13). *Isochrone in open source version of graphhopper?* Retrieved August 16, 2017, from https://discuss.graphhopper.com: https://discuss.graphhopper.com/t/isochrone-in-open-source-version-of-graphhopper/695/2

KARICH, P., & LEGRAIN, A. (2017, June 27). *GraphHopper Routing Engine*. Retrieved August 16, 2017, from https://github.com/graphhoppe: https://github.com/graphhopper/graphhopper

KONRAD, T. (2017, August 22). *Die OpenStreetMap-Gebäudeabdeckung in Österreich in Zahlen*. Retrieved August 22, 2017, from https://osm-austria-building-coverage.thomaskonrad.at: https://osm-austria-building-coverage.thomaskonrad.at

KREISER, K. (2016, November 16). *Isochrone demo*. Retrieved August 29, 2017, from https://github.com/valhalla: https://github.com/valhalla/demos/blob/gh-pages/isochrone/index.html

KREISER, K., & NESBITT, D. (2017, March 24). *SIF - Dynamic Costing within Valhalla*. Retrieved September 6, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla/blob/master/docs/sif/dynamic-costing.md

LANDESFEUERWEHRVERBAND BADEN-WÜRTTEMBERG. (2008). *Hinweise zur Leistungsfähigkeit der Feuerwehr.* Landesfeuerwehrverband Baden-Württemberg.

LANDESFEUERWEHRVERBAND HESSEN. (2015). *Hinweise und Empfehlungen zur Durchführung einer Bedarfs- und Entwicklungsplanung für den Brandschutz und die Allgemeine Hilfe der Städte und Gemeinden.* Landesfeuerwehrverband Hessen.

LINDEMANN, T. (2011). Rettungszeiten der Feurerwehr breim kritischen Wohnungsbrand. *BRANDSchutz - Deutsche Feuerwehr-Zeitung* , 946-952.

LUDWIG, I., VOSS, A., & KRAUSE-TRAUDES, M. (2011). *A Comparison of the Street Networks of Navteq.* Utrecht: International Conference on Geographic Information Science.

MACWRIGHT, T., CARRIERE, D., BORGHI, S., WINSEMIUS, R., & DEFOSSEZ, A. (2017, July 24). *turf/packages/turf-union/index.js.* Retrieved September 5, 2017, from https://github.com/Turfjs: https://github.com/Turfjs/turf/blob/master/packages/turf-union/index.js

MAPBOX. (2017, May 12). *Glossary: routing profile*. Retrieved July 26, 2017, from www.mapbox.com: https://www.mapbox.com/help/define-routing-profile

MAPZEN: ELEVATION. (2017). *Elevation service API reference*. Retrieved August 19, 2017, from https://mapzen.com: https://mapzen.com/documentation/elevation/elevation-service

MAPZEN: ISOCHRONE. (2017). *Isochrone service API reference*. Retrieved August 19, 2017, from https://mapzen.com: https://mapzen.com/documentation/mobility/isochrone/api-reference

MAPZEN: MULTIMODAL TRANSIT ROUTING. (2017). *Mapzen Multimodal Transit Routing*. Retrieved August 19, 2017, from https://mapzen.com: https://mapzen.com/projects/multimodal-transit-demo/?d=0&lat=40.7259&lng=-73.9805&z=12&c=multimodal&st_lat=40.744273&st_lng=-73.990328&st=Mapzen&end_lat=40.704651&end_lng=-73.9361&end=CartoDB&use_bus=0.5&use_rail=0.6&use_transfers=0.4&dt=2017-08-22T08%3A0

MCCAULEY, T., & GRI, L. (2017, May 19). *OpenRouteService*. Retrieved August 17, 2017, from https://github.com/GIScience/openrouteservice: https://github.com/GIScience/openrouteservice

MUEHRCKE, P. C. (1980). *Map Use - Reading, Analysis, and Interpretation* (2nd printing ed.). Madison, Wisconsion: JP Publications.

NEIS, P. (2010, August 28). *Etwas #OSM für die Feuerwehr*. Retrieved March 30, 2017, from http://neis-one.org: http://neis-one.org/2010/08/etwas-osm-fur-die-feuerwehr/

NESBITT, A. (2017, August 5). *osrm-isochrone release 3.0.0*. Retrieved August 17, 2017, from https://libraries.io: https://libraries.io/npm/osrm-isochrone/3.0.0

NESBITT, D. (2017, July 7). *How to add new profiles to valhalla?* Retrieved July 11, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla/issues/800

NESBITT, D. (2015, March 20). *Valhalla Open Source Routing*. Retrieved August 04, 2017, from https://mapzen.com/: https://mapzen.com/blog/valhalla-intro/

NESBITT, D. (2017, February 1). *valhalla/docs/thor*. Retrieved July 17, 2017, from Github valhalla-repository: https://github.com/valhalla/valhalla/blob/cf0a1353163ef713b5899518fadb6aca5ec3afc9/docs/thor/isochrones.md

NESBITT, D. (2017, February 1). *What are Isochrone Maps?* Retrieved August 19, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla/blob/master/docs/thor/isochrones.md

NESBITT, D., & WILTON, A. (2017, July 7). *THOR - DETERMINING THE BEST PATH*. Retrieved August 19, 2017, from https://github.com/valhalla: https://github.com/valhalla/valhalla/blob/master/docs/thor/path-algorithm.md

OPENROUTESERVICE: API FEATURES. (n.d.). *API Features*. Retrieved August 17, 2017, from https://developers.openrouteservice.org: https://developers.openrouteservice.org/portal/features

OPENROUTESERVICE: APIS. (n.d.). *API Catalogue*. Retrieved August 17, 2017, from https://developers.openrouteservice.org: https://developers.openrouteservice.org/portal/apis/

OPENROUTESERVICE: ISOCHRONES. (n.d.). *Isochronen.* Retrieved August 16, 2017, from https://openrouteservice.org: https://openrouteservice.org/reach?n1=47.341615&n2=8.506336&n3=12&a=47.369338,8.525054&b=0&i=0&j1=10&j2=10&k1=en-US&k2=km

OPENSTREETMAP GERMANY. (n.d.). *FAQ*. Retrieved August 17, 2017, from www.openstreetmap.de: https://www.openstreetmap.de/faq.html

OPENSTREETMAP WIKI: AMBULANCE STATIONS. (2016, May 11). *Tag:emergency=ambulance_station*. Retrieved August 31, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Tag:emergency%3Dambulance_station&oldid=1299190

OPENSTREETMAP WIKI: BORDER. (2017, July 3). *Tag:boundary=administrative.* Retrieved August 22, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Tag:boundary%3Dadministrative&oldid=1487373

OPENSTREETMAP WIKI: CITY BOUNDARIES OF GREAT BRITAIN IN OSM. (2017, April 22). *Ordnance Survey Opendata*. Retrieved August 25, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Ordnance_Survey_Opendata&oldid=1463673

OPENSTREETMAP WIKI: FIRE BOUNDARY. (2016, September 15). *Key:fire_boundary*. Retrieved August 31, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Key:fire_boundary&oldid=1344566

OPENSTREETMAP WIKI: FIRE STATIONS. (2016, December 3). *DE:Tag:amenity=fire_station*. Retrieved August 31, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=DE:Tag:amenity%3Dfire_station&oldid=1400938

OPENSTREETMAP WIKI: GRAPHHOPPER. (2017, June 9). *GraphHopper*. Retrieved August 16, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/wiki/GraphHopper

OPENSTREETMAP WIKI: ISOCHRONE. (2017, March 29). *Isochrone*. Retrieved June 20, 2017, from Openstreetmap Wiki: http://wiki.openstreetmap.org/w/index.php?title=Isochrone&oldid=1452891

OPENSTREETMAP WIKI: OPENROUTESERVICE. (2017, May 29). *OpenRouteService*. Retrieved August 17, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=OpenRouteService&oldid=1478898

OPENSTREETMAP WIKI: OPENROUTESERVICE TALK. (2014, December 1). *Talk:OpenRouteService*. Retrieved August 17, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Talk:OpenRouteService&oldid=1114199

OPENSTREETMAP WIKI: RESIDENTIAL. (2016, November 6). *DE:Tag:landuse=residential.* Retrieved Augst 22, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Tag:landuse%3Dresidential&oldid=1394404

OPENSTREETMAP WIKI: ROUTING. (2016, December 15). *Routing*. Retrieved March 24, 2017, from wiki.openstreetmap.org: http://wiki.openstreetmap.org/w/index.php?title=Routing&oldid=1409048

OPENSTREETMAP WIKI: ZOOM LEVELS. (2016, August 2016). *Zoom levels*. Retrieved March 30, 2017, from http://wiki.openstreetmap.org: http://wiki.openstreetmap.org/wiki/Zoom_levels

OPENSTREETMAP: COPYRIGHT AND LICENSE. (n.d.). *Copyright and License*. Retrieved June 29, 2017, from www.openstreetmap.org: http://www.openstreetmap.org/copyright/en

OXFORD DICTIONARY. (2017). *oxforddictionaries.* Retrieved March 03, 2017, from https://en.oxforddictionaries.com: https://en.oxforddictionaries.com/definition/emergency_services

PATEL, A. (2017, March 16). *Introduction to A**. Retrieved March 30, 2017, from http://theory.stanford.edu: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

PITTHAN, A. (2017, July 25). (B. Futterer, Interviewer)

PLATTNER, H. P. (2003). *Wir lassen die Feuerwehr im Dorf.* Retrieved March 28, 2017, from www.feuerwehr-balduinstein.del: http://www.feuerwehr-balduinstein.de/plaintext/sicherheitstipps/wirlassendiefeuerwehrimdorf/index.html

PORSCHE AG. (1978). *Feuerwehrsystem - O.R.B.I.T. - Entwicklung eines Systems zur Optimierten Rettung Brandbekämpfung mit Integrierter Technischer Hilfeleistung.*

PREVENT ING. (2017). *Bedarfsplanung*. Retrieved March 30, 2017, from https://preventing.de: https://preventing.de/bedarfsplanung/

RAMER, U. (1972, November). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing* , pp. 221–316.

RAMM, F. (2011, January 28). *OpenStreetMap Die freie WikiWeltkarte Übersicht – Chancen – Grenzen.* Retrieved August 22, 2017, from www.geofabrik.de: http://www.geofabrik.de/media/2011-01-28-dgfk-openstreetmap.pdf

RAMM, F. (2017, March 23). Routing-Engines für OpenStreetMap. Passau, Bavaria, Germany.

RAMM, F., & TOPF, J. (2010). *OpenStreetMap - Die freie Weltkarte nutzen und mitgestalten* (3. Auflage ed.). Berlin: Lehmanns Media.

RYLOV, M. (2015, February 26). *Major Update for OpenRouteService Backend*. Retrieved August 17, 2017, from http://k1z.blog.uni-heidelberg.de: http://k1z.blog.uni-heidelberg.de/2015/02/26/ors-new-backend/

SÄCHSISCHESES STAATSMINISTERIUM DES INNERN. (2005, December 01). Empfehlung des Sächsischen Staatsministeriums des Innern zum Brandschutzbedarfsplan. *Sächsisches Amtsblatt* , pp. 1168-1181.

SCHWARZE, B. (2005). *Erreichbarkeitsindikatoren in der Nahverkehrsplanung.* Dortmund: Institute of Spatial Planning of the Technical University of Dortmund.

SEDGEWICK, R. (1992). *Algorithmen.* Bonn: Addison-Wesley.

STAATLICHE FEUERWEHRSCHULE WÜRZBURG. (2016). *Feuerwehrfahrzeuge.* Würzburg: Staatliche Feuerwehrschule Würzburg.

STADT KARLSRUHE BRANDDIREKTION. (2006). *Brandschutzbedarfsplan der Stadt Karlsruhe.* Karlsruhe.

STALLMAN, R. M. (1986, February). What is the Free Software Foundation? *GNU'S BULLETIN* , p. 8.

SWAGGERHUB. (2017, July 24). *openrouteservice 4.1*. Retrieved August 17, 2017, from https://app.swaggerhub.com: https://app.swaggerhub.com/apis/OpenRouteService/ors-api/4.1

SYDEKUM, F. (2013, September 03). *Open Source Software und Free Software – Hintergründe, Ideologien und Vergleiche*. Retrieved August 10, 2017, from www.estrategy-magazin.de: https://www.estrategy-magazin.de/open-source-software-und-free-software-hintergruende-ideologien-und-vergleiche.html

THW. (n.d.). *www.thw.de/DE/THW/Bundesanstalt/Ehrenamt/ehrenamt_node.html*. Retrieved March 21, 2017, from THW - Technisches Hilfswerk: https://www.thw.de/DE/THW/Bundesanstalt/Ehrenamt/ehrenamt_node.html

TJCN. (2016, April 18). *Karamay City Statistical Communique on National Economic and Social Development 2015*. Retrieved March 31, 2017, from www.tjcn.org: http://www.tjcn.org/tjgb/201604/32814_2.html

VELDEN, L. (2014). *Kürzeste Wege in Graphen: Der Dijkstra-Algorithmus*. Retrieved March 30, 2017, from www-m9.ma.tum.de: http://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_de.html

VERSION EYE GMBH. (n.d.). *osrm-isochrone : 3.0.0*. Retrieved August 17, 2017, from www.versioneye.com: https://www.versioneye.com/nodejs/osrm-isochrone/3.0.0

WIKIPEDIA: CITY BOUNDARIES OF GERMANY IN OSM. (2016, February 5). *Wikipedia:WikiProjekt Kommunen und Landkreise in Deutschland/OSM*. Retrieved August 25, 2017, from https://de.wikipedia.org: https://de.wikipedia.org/w/index.php?title=Wikipedia:WikiProjekt_Kommunen_und_Landkreise_in_Deutschland/OSM&oldid=151120912

WIKIPEDIA: Douglas-Peucke. (2016, September 29). *Douglas-Peucker-Algorithmus*. Retrieved August 24, 2017, from https://de.wikipedia.org: https://de.wikipedia.org/w/index.php?title=Douglas-Peucker-Algorithmus&oldid=158329000

WIKIPEDIA: HILFSFRIST. (2017, January 23). *Hilfsfrist*. Retrieved March 23, 2017, from www.wikipedia.de: https://de.wikipedia.org/w/index.php?title=Hilfsfrist&oldid=161917081

WIKIPEDIA: OPEN SOURCE. (2017, August 07). *Open-source model*. Retrieved August 10, 2017, from https://en.wikipedia.org: https://en.wikipedia.org/w/index.php?title=Open-source_model&oldid=794348418

WIKIPEDIA: OPISOMETER. (2016, November 6). *Opisometer*. Retrieved March 30, 2017, from https://en.wikipedia.org: https://en.wikipedia.org/w/index.php?title=Opisometer&oldid=748120998

WIKIPEDIA:MAP SERVICE COMPARISON. (2017, August 17). *Comparison of web map services*. Retrieved August 24, 2017, from https://en.wikipedia.org: https://en.wikipedia.org/w/index.php?title=Comparison_of_web_map_services&oldid=795915447

WITT, W. (1970). *Thematische Kartographie - Methoden und Probleme, Tendenzen und Aufgaben* (2. Auflage ed.). Hannover: Gebrüder Jänecke Verlag.

WOLTER, M. (2017, July 31). (B. Futterer, Interviewer)

Appendix

i Glossary of specific words in the field of emergency services

To facilitate the comprehension and to avoid misunderstandings regarding the emergency service specific terms they are translated in the following glossary. For further translations in this field see the emergency service specific dictionaries at:

http://ff-eimsbuettel.de/index.htm?/siteinfo/infoserv/start.shtml
http://www.feuerwehr-aying.de/infos/deutsch-englisch.php
https://www.vfdb.de/lexikon.

Aerial Ladder Platform	Teleskopmast (TM), Hubarbeitsbühne (HAB)
alarm response area	Ausrückebereich
alerting of the emergency services	Alarmierung der Einsatzkräfte
application area	Einsatzbereich
arrival at the scene	Ankunft am Einsatzort
blue flashing light	Blaulicht
breathing apparatus unit	Gerätewagen-Atemschutz (GW-A)
career fire brigade	Berufsfeuerwehr (BF)
career fire fighter	Berufsfeuerwehrmann
chapter	Ortsverband (OV) des THW
civil protection	Zivilschutz, Bevölkerungsschutz, Katastrophenschutz (KatS)
Command Support Unit	Einsatzleitwagen (ELW)
Command Vehicle	Kommandowagen (KdoW)
critical apartment fire in an upper floor	kritischer Wohnungsbrand im Obergeschoß
customs	Zoll
degree of achievement	Erreichungsgrad
Development of a system to optimize rescue, fire fighting with the use of integrated technical assistance	Entwicklung eines Systems zur Optimierten Rettung, Brandbekämpfung mit Integrierter Technischen Hilfeleistung (ORBIT)
emergency	Notfall, Einsatz
Emergency Ambulance	Rettungswagen (RTW)
emergency call	Notruf
emergency drive	Einsatzfahrt
Emergency Medical Services	Rettungsdienst (RD)
Emergency Physician Car	Notarzteinsatzfahrzeug (NEF)
emergency service	Einsatzorganisation in Deutschland: Behörden und Organisationen mit Sicherheitsaufgaben (BOS)
emergency service dispatching	Disposition von Einsatzfahrzeugen
emergency vehicle	Einsatzfahrzeug
Equipment Vehicle	Gerätekraftwagen (GKW)

Federal Agency for Technical Relief	(Bundesanstalt) Technisches Hilfswerk (THW)
Federal Office of Civil Protection and Disaster Assistance	Bundesamt für Bevölkerungsschutz und Katastrophenhilfe (BBK)
fire brigade	Feuerwehr (FW)
Fire Engine	Löschfahrzeug (LF)
fire station	Feuerwehrgerätehaus
fire truck	Feuerwehrfahrzeug
first unit	ersteintreffende Einheit
functional strength	Funktionsstärke
governmental BOS	staatliche BOS
HazMat (Hazardous Materials) vehicle	Gerätewagen-Gefahrgut (GW-G)
Heavy Rescue Vehicle	Rüstwagen (RW)
incident	Vorfall, Ereignis
Mobile Intensive Care Unit	Intensivtransportwagen (ITW)
Multi-Purpose Vehicle	Mehrzweckkraftwagen (MzKW)
non-governmental BOS	nicht staatliche BOS
non-police BOS	nicht polizeiliche BOS
non-police emergency services	nicht polizeiliche Einsatzorganisationen
police	Polizei
police BOS	polizeiliche BOS
protection of the constitution	Verfassungsschutz
public-safety answering point	Einsatzleitstelle, Leitstelle
on patrol	auf Streife
operation	Einsatz
Operational Support Unit	Gerätewagen-Logistik (GW-L)
outbreak of a fire	Brandausbruch
Patient Transport Ambulance	Krankentransportwagen (KTW)
Personnel Carrier	Mannschaftstransportwagen (MTW)
public-safety answering point	Einsatzleitstelle
References to the capability of the fire brigades in Baden-Württemberg	Hinweise zur Leistungsfähigkeit der Feuerwehr in Baden-Württemberg
reporting time	Meldefrist
Rescue Helicopter	Rettungshubschrauber (RTH)
rescue operations	Rettungsmaßnahmen
response time	Hilfsfrist
scene	Einsatzort, Einsatzstelle
search and rescue of people	Suchen und Retten von Personen
sirens	Martinshorn, Folgetonhorn

special signs	Sondersignal
special vehicle	Sonderfahrzeug
Squad Truck	Mannschaftslastwagen (MLW)
State Fire Brigade Association	Landesfeuerwehrverband (LFV)
State Fire Brigade School	Landesfeuerwehrschule (LFS)
(non-police emergency service) station	(nicht polizeiliche(s)) Gerätehaus, Wache
standard incident	standardisierten Schadensereignis
succeeding unit	nachrückende (Einsatz-)Kraft/ Einheit
successful reanimation	erfolgreiche Reanimation
Swap Body Vehicle (also called Prime Mover)	Wechselladerfahrzeug (WLF)
talk and disposition time	Gesprächs- und Dispositionszeit
technical assistance	Technische Hilfeleistung (TH)
test drive	Testfahrt
THW federal association	THW Bundesverband
time for investigation and time until the first actions become effective	Eingreifzeit und Zeit bis die ersten Maßnahmen zu wirken beginnen
travel time	Anfahrtszeit
turnout	Ausrücken
turnout time	Ausrückezeit
Turntable Ladder	Drehleiter (DL)
volunteer	Freiwillige(r)
volunteer fire brigade	Freiwillige Feuerwehr (FF)
volunteer fire fighter	freiwilliger Feuerwehrmann
Water Tender	Tanklöschfahrzeug (TLF)
Working Group of the Chiefs of Career Fire Brigades	Arbeitsgemeinschaft der Leiter der Berufsfeuerwehren (AGBF)