FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Introduction Of OpenStreetMap For The Automatic Generation Of Destination Maps

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Felix Kendlbacher
Matrikelnummer 1128051

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Michael Birsak

Wien, 20. Oktober 2016

Felix Kendlbacher　　Michael Wimmer

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Introduction Of OpenStreetMap For The Automatic Generation Of Destination Maps

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Felix Kendlbacher
Registration Number 1128051

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Michael Birsak

Vienna, 20th October, 2016

Felix Kendlbacher Michael Wimmer

Erklärung zur Verfassung der Arbeit

Felix Kendlbacher
Marktgasse 14, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Oktober 2016

Felix Kendlbacher

Danksagung

Danke an meine Freunde und meine Familie!

Acknowledgements

Thank anyone reading this, my family, friends, and thank everyone I know, for it certainly is a pleasure knowing you.

Additional thanks go out to the Oxford comma, for life would be miserable without it.

Kurzfassung

Eine ”Destination Map” ist eine Landkarte, die allen Reisenden innerhalb der festgelegten Grenzen erlaubt, zu einem gemeinsamen Ziel zu fahren, ohne die Startpunkte der einzelnen Reisenden wissen zu müssen. Um dies zu erreichen werden zuerst alle wichtigen Straßen ausgewählt, die notwendig sind um das Ziel zu erreichen. Alle anderen Straßen werden aus dem Datensatz gelöscht, um die dargestellten klarer erkennbar zu machen. Dann werden die übrig gebliebenen Straßen vereinfacht, damit komplexe Straßenverläufe, die nicht zum Befahren des Straßennetzwerkes notwendig sind, vermieden werden. Die vereinfachten Daten werden dann in einem iterativen Prozess verzerrt, um eine bessere Darstellung des Straßennetzwerkes zu erhalten. Dabei werden kleinere Straßen vergrößert, um sie besser sichtbar zu machen. In diesem iterativen Prozess werden neue Darstellungen anhand mehrerer Kriterien bewertet. Sollte eine neue Darstellung besser abschneiden, als die vorherige, so wird diese neue Darstellung als Ausgangspunkt für weitere Verbesserungen hergenommen. Eine automatisierte Methode für die Erstellung solcher "Destination Maps"wurde in dieser Arbeit umgesetzt, anhand der Arbeit von Kopf et al. [KAB$^+$10], wobei Bemühungen gemacht wurden, den ursprünglichen Algorithmus zu verbessern.

Abstract

A destination map allows all travelers, within the given region of interest, to reach the same destination, no matter where exactly they start their journey at. For this purpose the important roads for traversing the road network are chosen, while the non-important roads are removed for clarity. These selected roads are then simplified to reduce unnecessary complexity, while maintaining the structure of the road network. The chosen data is then tweaked to increase the visibility of the small roads. During this process the layout is iteratively changed and evaluated according to certain aspects, and if a newly proposed layout performs better than the old one, that new one forms the basis for all future changes. In this work a method for automatically creating destination maps is implemented based on the algorithm proposed in the paper by Kopf et al. [KAB$^+$10], with efforts made to improve the original work.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**

2 Related Work And Navigational Background **5**
2.1 Navigational Background 7

3 Destination Map Creation **11**
3.1 User Input 12
3.2 Basic Architecture 12
3.3 Obtaining Data 13
3.4 Selection Of Roads 13
3.5 Simplify Data 17
3.6 Data Layout 22
3.7 Rendering 29

4 Implementation And Used Technologies **31**
4.1 Open Street Map - OSM 31
4.2 Custom Data Format 34
4.3 cURLpp 34
4.4 TinyXML-2 34
4.5 Mapnik 34
4.6 Java OpenStreetMap Editor 35

5 Results And Suggested Improvements **37**
5.1 Changes To The Original Algorithm 39
5.2 Performance 39
5.3 Possible Improvements 41

Bibliography **47**

CHAPTER 1

Introduction

When numerous people are to travel to the same location, the creation of a map all people of the group can use to successfully navigate is not a trivial task. There are plenty of options for simply obtaining a map containing the whole area, thus allowing the whole group to successfully reach their destination, but these maps are too detailed for easy navigation, containing unnecessary small roads in areas none of the travelers will reach.

As can be seen in Figure 1.1, maps containing all the roads of the area of interest are not easily traversable. Showing all the possible ways does not aid navigation, as it is hard to identify and follow the interesting roads in these kind of maps. Furthermore some of the complexity of the road network is not necessary for navigation purposes, e.g. most ramps need not be depicted if navigation is the task.

Route maps, on the other hand, usually only connect two points. This kind of map is easier to navigate, but it needs to be created for every person of the group individually. Additionally, the issue with parts of the route being too small to be easily readable still persists, if the depicted area is too large. The problem is that these maps use the same scale for all the roads in the map, even though a closer view of some areas may be helpful in guiding the user to the destination.

In Figure 1.2 the big streets are easily identifiable at first glance. However smaller roads, close to the start and endpoint, are tough to navigate without a closer look. This problem is magnified with a larger area of interest, as the smaller roads are rendered even smaller.

Destination maps aim to solve these issues by containing all the necessary information for reaching the destination from a multitude of starting points, without cluttering it with streets which are not needed for the given goal, while magnifying the smaller roads to increase readability. For this, all important roads around and leading to the destination are put in a map, featuring only the essential information. Additionally small roads are blown up so they are navigable even when printed next to the motorways leading into the area of interest. This eliminates the common issue with choosing a scale for a map,

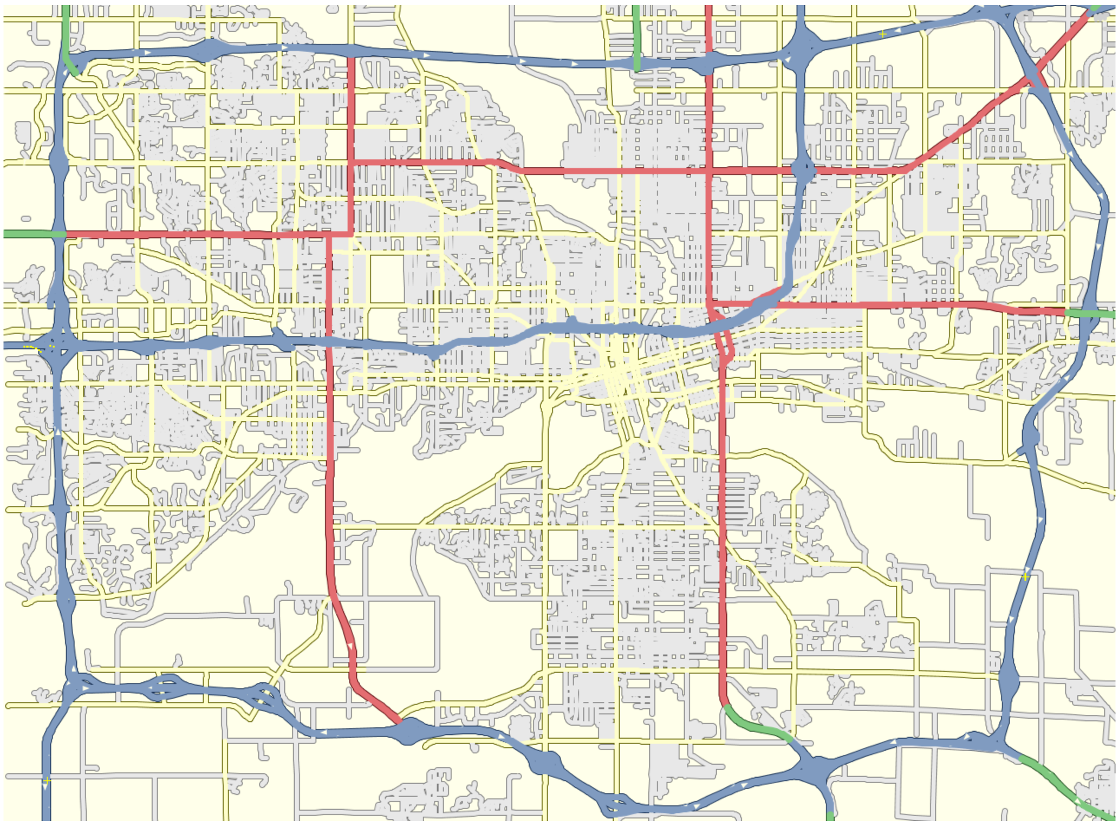


Figure 1.1: A map containing all the roads of the selected area. Many small roads are barely distinguishable from one another, others completely overlap since the scale of the map is too large to depict these roads without any overlap.

for which either the important roads of the surrounding area are visible, or the small roads in direct proximity to the goal. Rarely both types of roads are visible at a glance when normal maps are used.

Figure 1.3 shows such a destination map. Non-important roads were deleted from the map and the surroundings of the destination have been blown up to increase their visibility. Note that, despite many simplifications, sharp bends are still visible in the map.

As with hand-drawn destination maps, automatically created ones distort the original map data in order to simplify the navigation process. The accuracy of the depicted information is thereby reduced in order to increase the readability of the given area. Only information that is not vital to navigating the map is removed and special measures are taken in order to provide contextual information despite the simplifications. This work is about an algorithm to automatically create these types of destination maps, which is based on the paper written by Kopf et al. [KAB^{+}10].

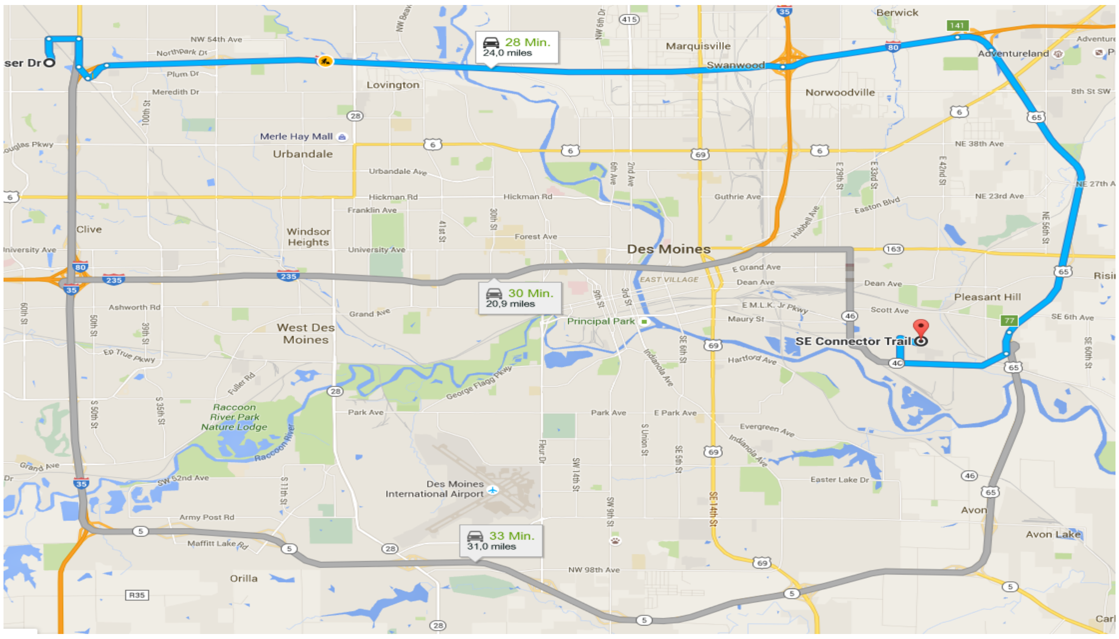


Figure 1.2: A typical route map, generated by google maps. The blue route is the proposed one.

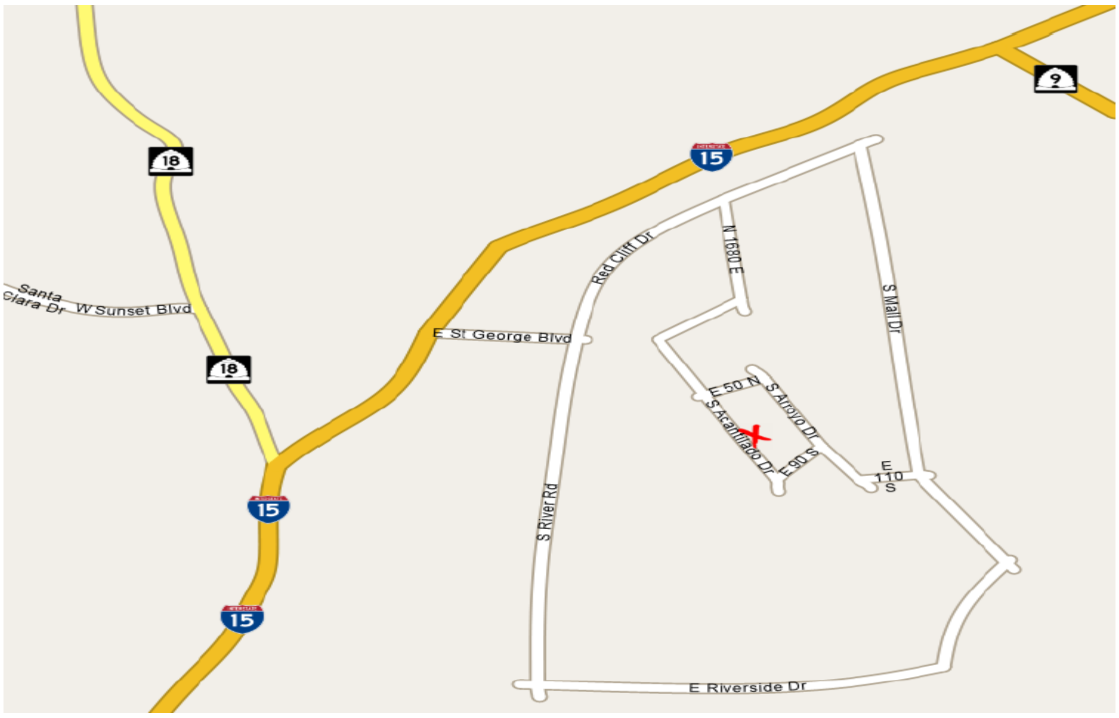


Figure 1.3: Automatically generated destination map from the work by Kopf et al. [KAB^{+}10]. The destination is marked with a red cross.

CHAPTER 2

Related Work And Navigational Background

This work is an implementation of the algorithm originally proposed by Kopf et al. [KAB+10] that tries to improve it in some aspects, all of which are specifically denoted as deviation from the original algorithm. Kopf et al. propose an automated method for generating destination maps, which are maps that allow navigation to a single destination from an arbitrary starting point outside of a user-defined border (It is only navigable for some starting points within the border). For this the predicted routes to the destination are computed and rings of streets of a given street-type around the destination are built to allow for easy navigation (see Section 3.4). To increase readability the data is simplified (Section 3.5) and the layout of the road network is distorted (Section 3.6). The algorithm is described in greater detail later on in this paper.

The work of Karnick et al. [KCJ+10] uses an alternative way of displaying route maps, by using detail lenses that zoom into important points of interest of the route displayed, see Figure 2.1. These lenses are placed alongside the overview of the route, thus enabling the driver to have both an overview of the area as well as the necessary intersection details shown at the same time. Instead of the approach used in the algorithm by Kopf et al., Karnick et al. do not distort the underlying data, but show different zoom levels on the same map. This idea may well be used for destination maps, not only for route maps as in the original paper. Issues with using detail lenses in destination maps is the problem of where to place the lenses and a potentially high amount of points of interest to be displayed. The placement issue is worse in destination maps, for they not only show a single route, but a lot of the surrounding area of the destination. However in most cases empty spaces can be found on destination maps, thus an approach that replaces the road layout process of the algorithm with placement of detail lenses may be successful in improving the outcome.

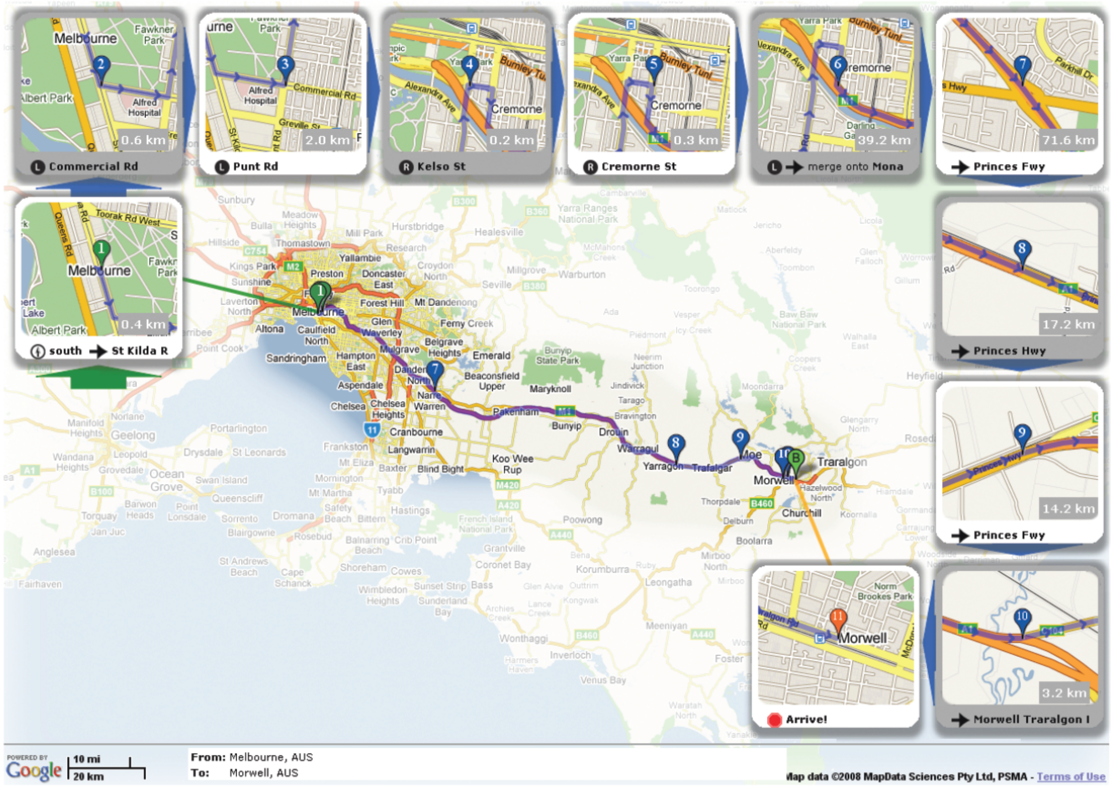


Figure 2.1: Result of the algorithm proposed by Karnick et al. [KCJ$^+$10]. The Figure was taken from said paper. The highlighted intersections are placed on the border of the map, so they do not overlap with the route itself. The route contains markers that identify which detail lens corresponds to the given intersection. The detail lenses are additionally drawn in order of appearance to make finding the correct lens easier for travelers.

Grabler et al. [GASP08] create a tourist map containing a simplification of the original map data, in a typical top-down view. The map is augmented with representations of points of interests in the area, shown as three-dimensional models placed into the map at their respective positions, see Figure 2.2. The list of interesting points is based on data obtained from the web, which is further analyzed to choose buildings which are likely to be used as navigational aid. These points are then ranked according to input interests of the user, which prevents cluttering of the map with unnecessary information. In a similar fashion the to-be-retained roads are computed, with special consideration for navigating the points of interest chosen by the user.

The combination of tourist maps and detail lenses is presented by Birsak et al. [BMWW14]. For this system the detail lenses contain at least one point of interest and its surroundings in greater detail than in the overview, and additional information, e.g. the name and address of the contained points of interest. These points of interest are obtained from a

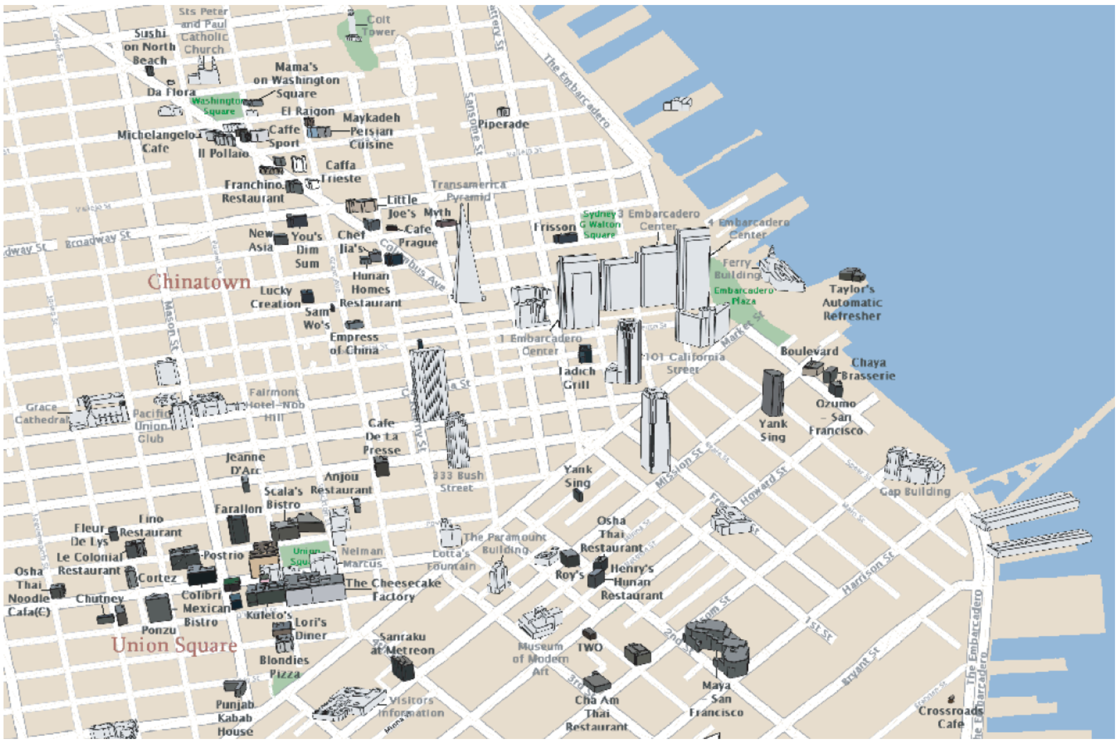


Figure 2.2: Exemplary result from the algorithm by Grabler et al. [GASP08]. The image is taken from that paper. The road network is simplified in only to a certain degree by omitting a small subset of roads. Tourist attractions are rendered to be easily recognizable for tourists.

web-source. Similar to the algorithm implemented in this work, one goal is to connect the points of interest in a manner that is close to optimal and easy-to-follow at the same time, thus some shortest ways are omitted in order to improve navigability. The overview map additionally marks the surroundings of the points of interest, which are covered in the detail lenses, see Figure 2.3.

2.1 Navigational Background

Human beings use local references for navigating, not exact values or maps. As proposed by Golledge [Gol99] "Virtually no one knows the latitude and longitude of even the best-known features in their environment". The fact that we instead use relational references as aids during navigation enables us to create a map which does not stick to the exact, real-world coordinates, but is still helpful with navigational tasks, possibly even more than an exactly made map. This knowledge allows us to distort and simplify data, while maintaining its navigability.

Since people perceive road structures hierarchically, they first recognize the important,

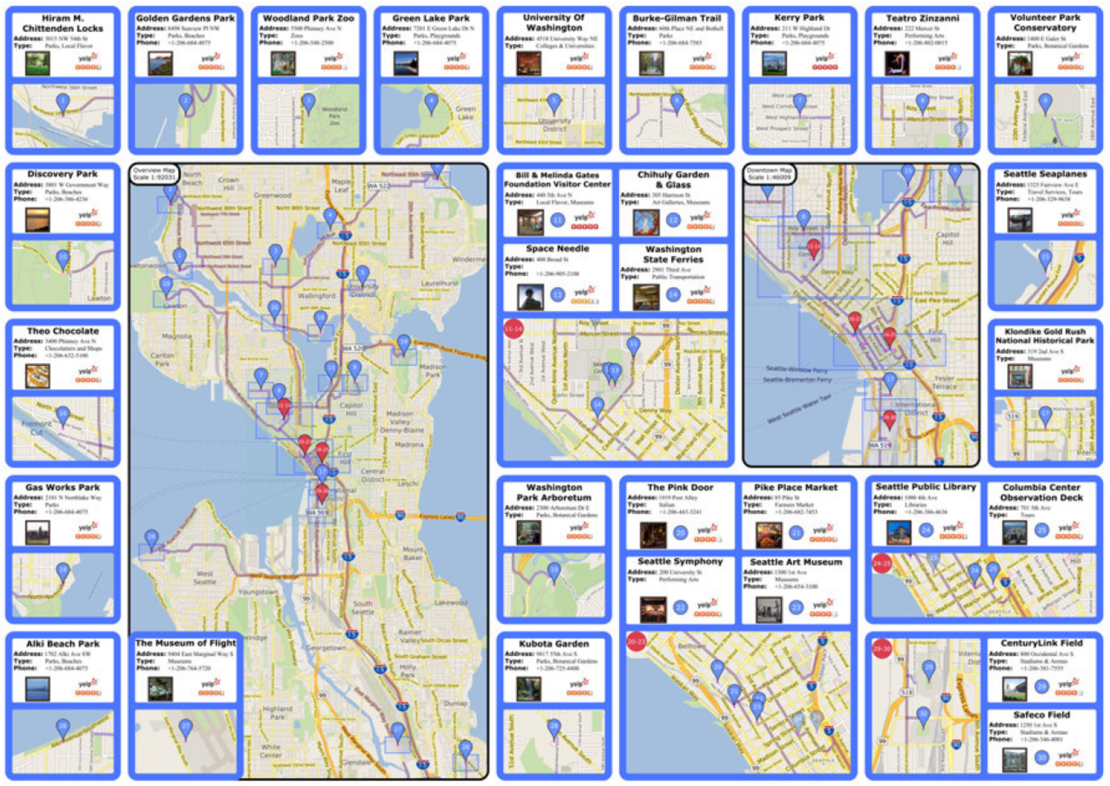


Figure 2.3: Image taken from Birsak et al. [BMWW14]. Their work uses the previously introduced detail lenses to allow for easy navigation to the specified points of interest, while portraying additional information about the given attractions in the lenses. The red markers depict regions with multiple points of interest. These regions are shown in an enlarged version to aid navigation.

big roads. Smaller roads are perceived successively only when they are needed to reach the destination [KAB$^+$10]. The roads chosen to be displayed in the map are considered in a hierarchical manner as well. To this end hierarchical rings are built around the destination, each ring containing roads of a specific street-type. In order to decrease the complexity of the resulting road-network, paths along these hierarchical rings are discounted when connecting streets to the destination, thus favoring paths that take the already considered rings.

In order to improve navigability, contextual information is kept at junctions, even if some of the streets at the intersections are not used as roads to travel along. The retained context makes reading the map at turns easier, by showing all the streets connected to the given junction. Additionally, the contextual information is not large enough to clutter the map unnecessarily.

After selecting a set of roads to be displayed in the map, we then need to present it in an easily readable manner. This task of making map-data clearer "is accomplished by performing three types of generalization on the route: (1) the lengths of roads are distorted, (2) the angles at turning points are altered, and (3) the shapes of the individual roads are simplified" [AS01]. All three generalizations are used in the work by Kopf et al. to improve the data. The first point emphasizes short road segments, the second point deemphasizes only slight changes of direction, while the third point allows an easier reading of the map. Furthermore streets deemed unimportant for navigating to that specific destination are removed.

CHAPTER 3

Destination Map Creation

I implemented the algorithm proposed in the Automatic Generation Of Destination Maps paper by Kopf et al. [KAB^{+}10]. There have been numerous small adjustments made to the original algorithm, which are explicitly marked as such in the following text.

The algorithm consists of multiple parts, which are described in greater detail below. As first step, the location of the destination, along with the desired area of interest is provided as input by the user. The pipeline executing the algorithm is then started by downloading the street data. From this data a subset of important roads is selected and used in the later steps.

For selecting the important roads, hierarchical ring roads around the destination are computed. All the nodes of these ring roads are then connected to the destination with a slightly modified version of Dijkstra's shortest path algorithm which aims to strike a balance between including the fastest roads while keeping the resulting road network simple and easy to navigate. Roundabouts that are only partially present in the road network are then added to the network as a whole. All roads that are now part of the network are then extended outwards from both endpoints and small segments of non-selected roads at junctions are added to the road network as well, in order to provide context which exit the user has to take.

All roads not selected are then discarded, and the remaining data is simplified. In a first step all roundabouts are replaced by simple intersections and dead ends at previously extended roads are removed until an intersection is reached. Then parallel roads with the same street name are merged into a single street, this mostly concerns motorways. Ramps connecting two roads are additionally replaced with simple intersections. If a part of the street network was simply taken in order to turn around on motorways, that part is removed as well.

Then intermediate nodes, i.e. nodes that do not represent junctions or dead-ends, are removed if they do not represent a sharp bend in the dataset. All remaining edges are then clipped against the input area of interest. The layout of the simplified data is optimized during the data layout step, and finally forwarded to the renderer.

3.1 User Input

The basic, necessary user input consists of latitude and longitude coordinates of the destination node, as well as the area of interest to be portrayed in the final outcome. Additionally, the user has to provide the physical dimensions (e.g. DIN A4) of the destination map that is provided as the final output of the system. It is important to note that, for an undistorted output, the ratio between the output height and width has to be similar to the ratio of the area of interest, thus the user interface automatically adjusts it.

Beyond these obligatory parameters, the user can adjust multiple advanced options. All weights of the layout step are adjustable to allow for experimentation during that process. Additional options for loading a data file from the local computer (as opposed to the server), adjusting whether the experimental parts of the algorithm are to be performed are provided.

In addition some pre-made settings are selectable in order to quickly test the algorithm. These pre-made settings set all the necessary values in order for the algorithm to operate on that given area. Some adjustments to be kept over multiple instances of executing the program are available as well. These include the server to be reached for the data and the path to the rendering style-sheet.

3.2 Basic Architecture

The algorithm consists of many pipeline parts, all of which perform a specific action on the dataset. The pipeline can easily be modified or extended to omit certain optimization stages or to add a new method to increase the quality of the output. The pipeline is started from the graphical user interface controller and runs in its own thread to keep the user interface responsive, thus enabling status messages about the ongoing progress.

The OSM data is represented in our own data structure during the processing of the algorithm, but is transformed back into basic OSM data after all optimizing steps have been taken. Furthermore, this transformation step is done at set points in the pipeline, where a snapshot of that data is printed as OSM file. This resulting data is then used to create an image via Mapnik.

All options to be set by the user are stored in an option file object, which is a plain container for values to be checked by the algorithm. As an example, this contains the destination and the bounding box, limiting the area of interest. This option file is consulted by each part of the algorithm individually for the necessary values, which are set once just before the pipeline is started.

3.3 Obtaining Data

The necessary data is obtained using the OSM extended API (Xapi). The Xapi-requests are made within the given bounding box (provided as input by the user), where each request is made for all ways and nodes of a specific street-type, starting with the most prominent one, motorways, and moving down to the smallest one. This reduces the payload and prevents invalid responses in case the amount of data is too large. The information gained is then parsed into the custom data format and all the datasets with differing street-types are added up to form one big dataset with all the necessary information.

Even though it is not programmed that way, this process lends itself to parallelization, since during parsing one street-type it is easily possible to start the request for data of the next street-type. This might yield a good improvement in performance if the depicted area, and thus the corresponding dataset, is very large. Alternatively, the dataset may be loaded from an OSM file. This file can contain all the street-types and all of the data will be parsed into the custom data format at once.

After obtaining all the data, the closest node to the entered coordinates for the destination is determined. For this a comparison of the positions of the nodes and the input destination-position is done, and the closest node is set as destination node.

One of the issues is that the Xapi service refuses to send a valid response if the resulting dataset is too big. This behavior usually appears when requesting small street-types in a large area. My solution for solving this, is to simply shrink the area of interest around our destination, making sure that the destination node still is part of the obtained data, while removing some of the small streets at the border. The shrinking amount depends on the total size of the area of interest, this process will be repeated a few times if the requested amount of data is still too large. While there might be a case where an important connection, which would be nice to have in the resulting destination map, is lost through this process, we found it to be a sufficiently effective solution for the present technical limitations. Since the issue only occurs with small roads, the observation that the result is still effective corresponds to the notion of hierarchical navigation as expressed by Kopf et al. [KAB+10]. This is because the omitted small roads are located at the border, where more prominent roads are used for navigation. Note that the limitations do not exist if all the data is stored locally and not downloaded on demand.

3.4 Selection Of Roads

This process identifies the important roads for reaching the destination and removes all other edges and nodes from the dataset. The remaining subset of only the to-be-traversed roads is then passed on to the simplify data step. The selection of roads to be displayed in the destination map is done by first creating the aforementioned hierarchical rings around the destination, which serve as basic structure to travel upon. These rings are then connected to the destination node and all encountered roundabouts are considered.

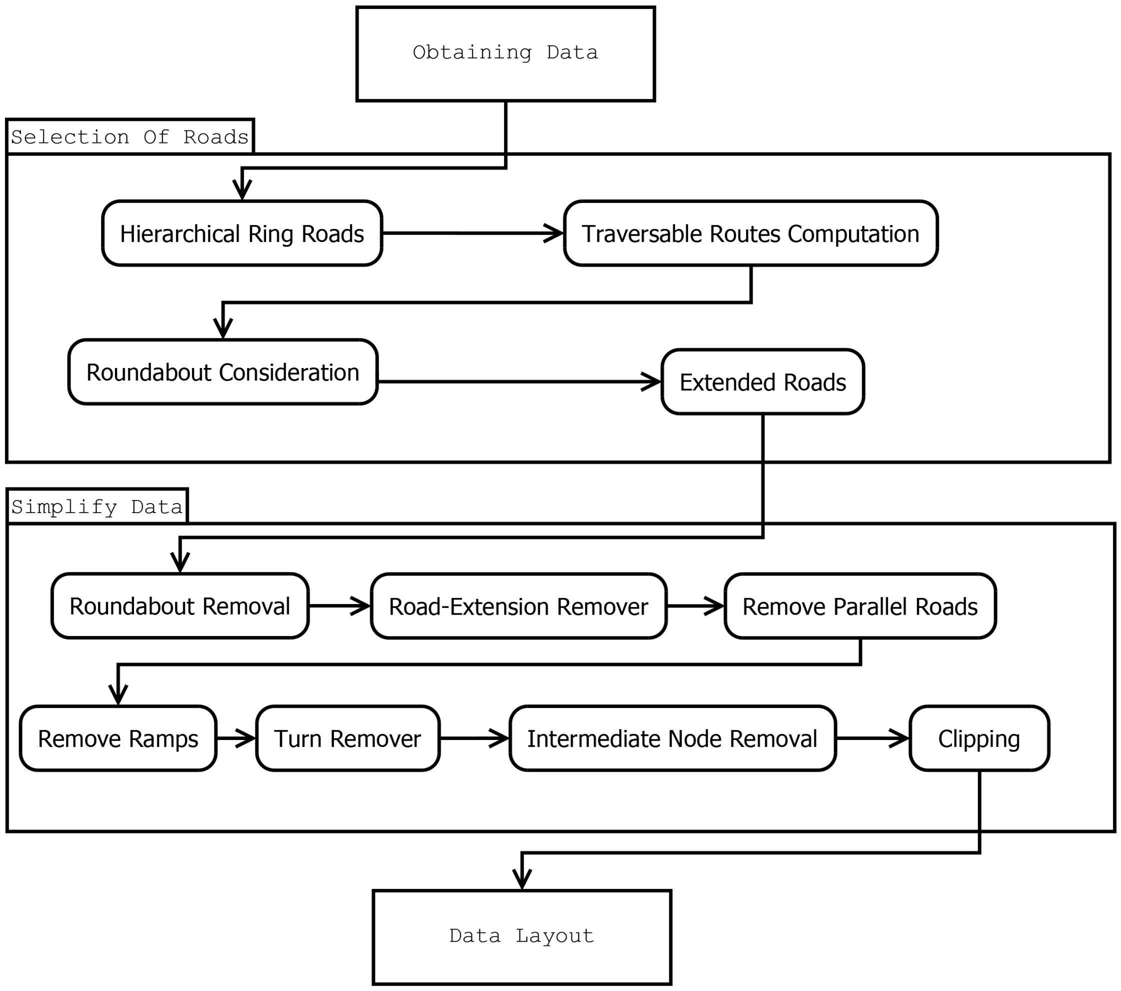


Figure 3.1: Diagram depicting the components used during the selection and simplification steps of this algorithm and the order in which they are applied. Diagram was made using the Dia diagram editor [Mac16].

Lastly more context is added to the layout by extending the dead-ends of all roads that are currently part of the selection.

3.4.1 Hierarchical Ring Roads

The selection of the hierarchical ring roads is performed by sampling the dataset in at least eight directions with regards to the destination (by default 48 samples are taken, this however depends on the user's input). For the samples a half-line is created, which starts at the destination node and aims in one of the directions. All edges that have an intersection with that half-line are potential samples. The taken sample is always the one closest to the destination node, with bigger roads blocking smaller roads. Of the eight

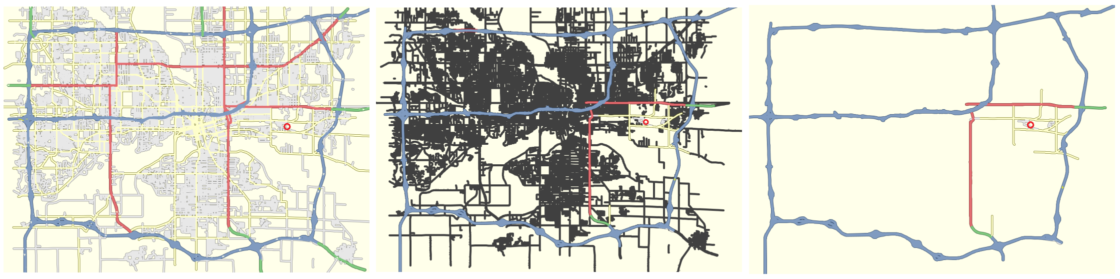


Figure 3.2: Exemplary case of the work done by the selection of roads part of the algorithm. The left picture shows the whole dataset, used as input. The picture in the center shows both the selected roads and the grayed out non-taken roads. The right picture shows only the remaining roads, as they are passed on to the simplification step. The red circle marks the location of the input destination.

minimum directions four aim at the corners of the area of interest, while two are placed horizontally (to the left and right side of the input area) and the last two are placed vertically (to the top and bottom side of the input area). All other samples are placed equidistantly in between the eight aforementioned sample directions.

Samples are taken for each relevant street-type in descending order with regards to the importance of the corresponding street-type, first sampling the motorways, then the trunks and so forth. A single sample consists of exactly one edge, which is extended in both directions, until the next node with valency unequal 2 is reached (i.e. until the next junction or dead-end). The obtained samples are sorted with regards to the direction in which they were obtained.

Neighboring samples, obtained from neighboring sample directions, are then connected if there is not more than one empty sample in between them (an empty sample exists at sample directions where no edge of the given street-type intersected the half-line). This is done by using one of three different methods. In case both samples lie on the same street they are simply connected by adding the edges along that street and in between the two samples to the selection. If the streets corresponding to the samples have a mutual intersection with one another, then both edges are extended up to that intersection. If none of the two cases apply, a connection is made with a standard Dijkstra algorithm. If, however, that connection is more than twice as long as the direct beeline between the two samples, the computed connection is not added to the ring roads. These created hierarchical ring roads build the basis for navigating the dataset.

In addition to these edges, all roads belonging to the highest present street-type, as well as their corresponding link roads, are automatically considered as ring roads and taken into account for the upcoming steps. This is done because most travelers enter the area of interest via a street of the highest present street-type. The corresponding link roads are added as well to maintain the connectivity of the streets.

Path Search

For all tasks that need a connection between two edges or nodes we implemented the well-known Dijkstra algorithm. It operates on a weighted graph, with weights being non-negative, and finds the shortest paths from one vertex to all other vertices. The Dijkstra vertices are the nodes of the dataset with a valency unequal 2, since intermediate nodes do not influence the topology of the graph. The exception to that rule is if the starting or destination node is of valency two, in which case nodes with valency 2 may exist within our graph. Each edge is assigned a weight, which is proportional to the traveling time until the next Dijkstra vertex is reached (not counting intermediate nodes). The traveling time is computed by dividing the spanned distance by the maximum speed allowed on that given edge. For the algorithm to work, the time needed to reach every given node is saved, and all nodes are initialized with a value of infinity, except for the starting node, which is initialized with zero.

The starting node is then put into a queue, which is sorted according to the time needed to reach that vertex, and then the algorithm loops for as long as there is a node in the queue. For each node in the queue all neighboring nodes are analyzed. If a faster way is found to a vertex, the new time is saved and checked whether the vertex needs to be inserted into the queue. There are three cases:

1. If the vertex has never been in the queue, it is added together with the time traveled.
2. If the vertex is currently in the queue, the old entry is removed and the vertex is reinserted with the new traveling time to maintain the descending order. (the new traveling time is automatically shorter than the old one, otherwise the vertex's time would not have been saved and no check would have been made)
3. If the vertex has been in the queue, but has been already visited, and is thus no longer in the queue, it is not re-inserted, but simply ignored.

Additionally, the last edge traversed in order to reach that vertex is saved, which allows to recompute the path taken to that specific node.

If a destination node is specified as well as a starting node, the algorithm has an additional termination condition, which triggers if no shorter path can be found to that goal node, i.e. if the time traveled to the goal node is shorter than the shortest distance found in the queue of yet unvisited vertices. This is used if two samples of the hierarchical ring roads are to be connected. If no goal node is specified the data containing the traveled times to the vertices is returned together with the information which edge was taken to get to that vertex.

3.4.2 Traversable Routes

After finding the hierarchical ring roads, they still need to be connected to our destination node. For this the shortest paths are computed, with the above described Dijkstra path search, from the destination node. All nodes that are part of a ring road and have a valency unequal 2 are connected to the destination node and all edges along the way are added to the selection. In order to prevent the resulting paths from becoming too complicated or cluttering the map, traveling along ring roads is discounted by 70 percent and taken turns are penalized with a fixed cost of ten seconds during this execution of the Dijkstra algorithm, as originally proposed by Winter [Win02].

3.4.3 Roundabout Consideration

In order to improve the original algorithm [KAB$^+$10], consideration for traversed roundabouts has been added. After selecting the paths to our destination, all roundabouts, which are at least partially selected, are further evaluated (i.e. if one or more nodes of these paths are tagged as roundabout, the whole roundabout is considered). If there are more than two roads entering or leaving the roundabout, all adjacent edges are used in the layout. Additionally, if there are exactly two roads entering or leaving the roundabout, but they belong to different streets (i.e. a turn is made) all leaving edges are put into the layout as well. In the other cases no context is added to the dataset, for the roundabout is simply crossed (i.e. the traveled street remains the same). All roads, which are added to the dataset for context, are extended one edge beyond their first node with valency unequal 2. That additional context helps the travelers to identify the correct exit to take.

3.4.4 Extended Roads

To provide context information for the users, all selected edges are followed outwards from both the starting and the endpoint of that edge, unless the next edge at that point was already selected from a previous step. The edges that do have an unselected, adjacent edge of the same street are extended until the road has either been extended for 1.5 kilometers or until the road name or street-type changes. This provides greatly extended road segments that are in a later step reduced, so they provide context without cluttering the map.

3.5 Simplify Data

The simplification process is a very important part of this algorithm as it fulfills two goals at once. It makes the map easier to read by removing unnecessary complexity from the road network, for example if a road segment shows a slight curvature there is not much information lost if it is portrayed as being straight. Additionally it reduces the complexity of the data which is important for the computing-intensive procedure of creating many different versions of the data-layout, in order to optimize it. Before any of the described processes are started, all edges and nodes that are not part of the selected road network (as made by the steps for road selection) are removed from the dataset. All following operations are only executed on the remaining important roads.

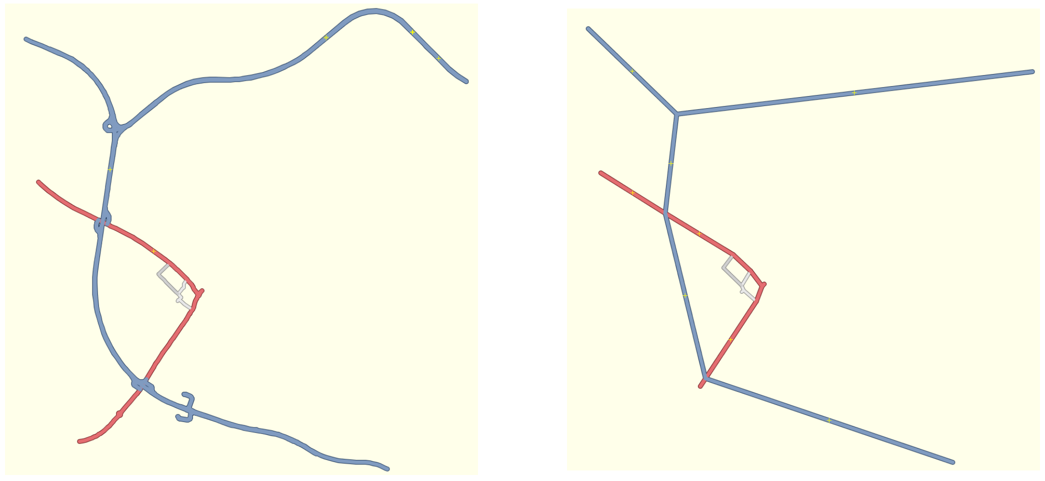


Figure 3.3: Situation before and after the simplification step.

The simplification happens in multiple steps. First the roundabouts are simplified into a single intersection, then the earlier extended roads are shortened again so they become short tails differentiating T or X junctions. Then parallel roads of the same name are merged into a single road and all link roads are replaced by normal intersections. The streets, which were only taken in order to enable turning around on motorway streets, are removed if they are not connected to any other part of the road network and all intermediate nodes that do not contribute much to the overall network are deleted. In a final step the data is clipped against the borders input by the user so all nodes and edges are inside these borders.

3.5.1 Roundabout Removal

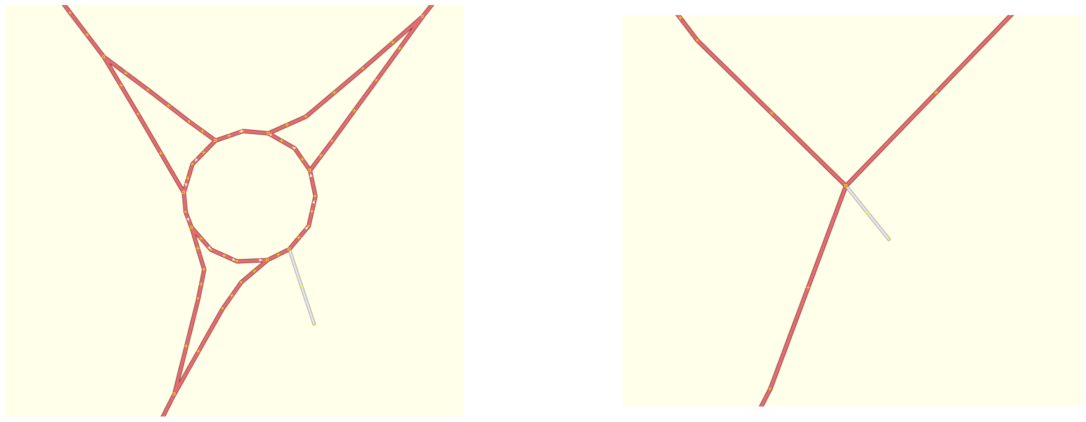


Figure 3.4: A typical case of roundabout simplification. The left image depicts a roundabout before the simplification step. The right image shows the same roundabout after it was simplified.

The second part of treating roundabouts, which is not used in the paper this work is based on [KAB$^+$10], is designed to simplify roundabouts, since they often produced unwanted results by cluttering the map and introducing unnecessary complexity to the layout. In a first step a new center node is created by averaging all the roundabout nodes' positions. Then all edges of the roundabout are deleted, as well as all nodes of the roundabout that are not entering or leaving points of that roundabout. Thus only the newly placed center node and the nodes that represent exit points of the roundabout remain. These remaining roundabout nodes are then connected to the new center, thereby simplifying the roundabout to a simple intersection.

Secondly, a check is made whether two or more edges, which share a name and street-type, end up at the same node with valency unequal 2 when traversed starting at our new center node. If that is the case all edges in between these two nodes are deleted and replaced by a single edge, directly connecting the center and the node where the edges met. This is necessary for roads often split up into two one-way roads before encountering a roundabout, one for entering it and one for leaving it, which unnecessarily clutters the map without conveying any useful information. An example for the obtained result can be seen in Figure 3.4.

3.5.2 Road-Extension Remover

All nodes with valency 1 and their incident edges, which were earlier extended, are followed and removed, until the edge to be deleted is connected to another street (i.e. until a node with valency greater than 2 is reached). The exception being ring roads, which are not shortened, even if they have a node with valency 1. The small dead-end kept at every road is there in order to provide context about what kind of junction is present at the given node, as it can be helpful to know that a specific intersection is a T or X junction.

3.5.3 Remove Parallel Roads

In this step all nodes that are considered for the merging of parallel roads, which are all nodes that have a valency other than 2 or nodes with valency of 2 where the road name changes at that node, are selected. (i.e. a node where the two connected edges have different names). All link roads, however, are not considered for merging since they are treated separately in the ramp removal step. Then all selected nodes are evaluated for parallel streets with the same name as the ones connected to the node and within a distance of, at most, 200 meters. If such a parallel edge is found, it is split at the point closest to the node. The new node splitting the edge is then merged with the original node. The resulting merged node has its position at the average of the two other nodes in this implementation, since the one by Kopf et al. [KAB$^+$10] does not specify where to place the new node.

For each merged node that is connected by more than one set of edges to another node with valency unequal 2, all of the edges connecting these two nodes are deleted, until only one connection remains between them. The retained path is the one of the highest street-type, or, if that is not applicable, the shortest available path. During this process link roads are not counted toward the valency of the nodes, since they are removed in the next step anyway. If a link road is present at one of the removed edges, the link road is adjusted to lead to one of the two connected nodes. The connection is made to the node that is closer to the loose end of the link road. The special treatment and re-linking of link roads is not described in the original paper [KAB^{+}10], but is consistent with the shown results.

3.5.4 Remove Ramps

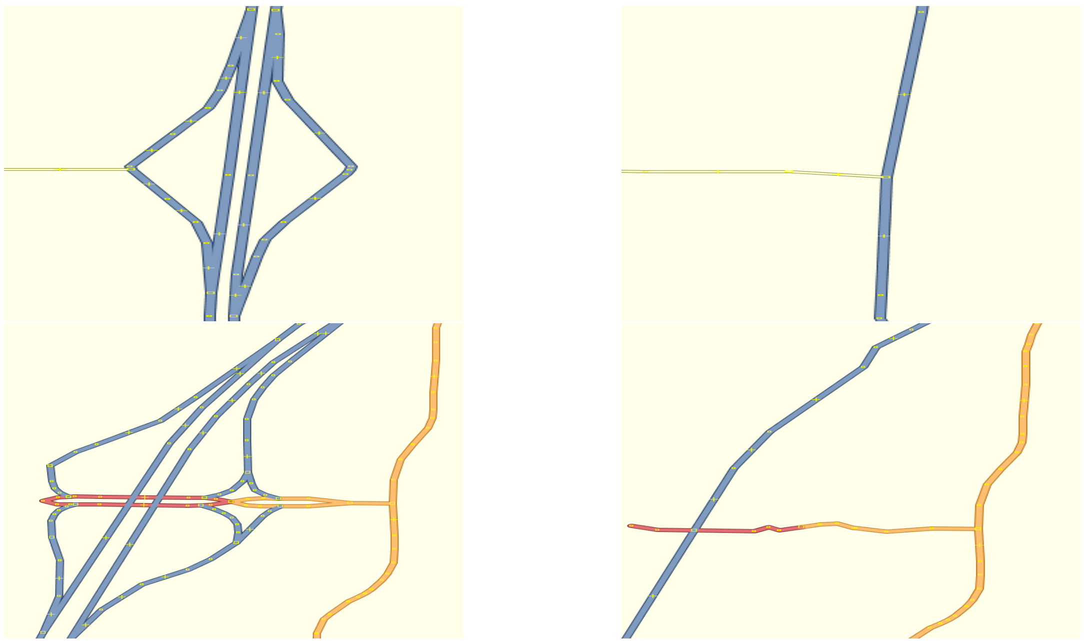


Figure 3.5: Two cases of merging parallel roads and removing ramps. The images on the left side portray the intersections before any simplifications, while the ones on the right display the simplified intersections.

All ramps connecting streets are removed and replaced by simple intersections, further increasing the simplicity of the acquired data. For this each edge with a link street-type (i.e. motorway link, trunk link, primary link, secondary link and tertiary link) is followed outwards in both directions until a node is found that is representing a junction with at least one connected edge not being a link road. The two roads belonging to the found nodes are then connected via a simple intersection. After all start and end node pairs are found, all link roads are deleted. Since the original paper [KAB^{+}10] does not go into detail about how to replace these ramps, the method used here may differ from the one used in the paper.

As first step a check is made whether the roads at the two nodes intersect at another point, not necessarily with a junction node, but simply if their edges overlap. For this the streets are followed outwards from the two nodes and checked for intersections between the edges. If such an intersection is found, an intersection node is inserted at that point.

If no intersecting pair of edges is found, two sets of neighboring nodes of the start and end node are built, and looked at for where a connecting edge has the shortest possible distance. The edge with the shortest distance is then taken as connection between the two nodes, which is viable for it is not necessary to exactly preserve where the link roads lead or started, as long as the connectivity remains. This is because the map is distorted during later simplification steps and during the data layout step, thus there is no visible difference between the correct and the approximated solution. If that shortest edge is however already part of the dataset, for example because there is more than one ramp connecting the two streets, no further actions are taken.

3.5.5 Turn Remover

In this step streets are considered, which are only part of the road network so travelers can change the one-way lane they travel on. For this all motorway nodes with a valency of 3 or higher are taken into consideration. First off, some neighboring nodes of the selected node are locked and marked as invalid for the upcoming task. Only nodes that are on the same motorway as the selected node are considered as invalid neighboring nodes.

Then all edges leading from that selected node are followed and their traveling distance is summed up. An edge is no longer followed, if it leads to one of the neighboring nodes, which were previously marked as invalid. If the sum of distances of the followed edges reaches one kilometer the edges stay in the road network. If, however, there are no more edges to traverse before the minimal distance of one kilometer is reached, all followed edges and nodes are removed, since they represent a part of the network that only exists as help for changing the direction on motorways. This process reduces clutter, if the turns on motorways are not solely done on roads that are marked as link roads.

3.5.6 Intermediate Node Removal

The step of intermediate node removal aims to reduce the complexity of roads without taking away information necessary for navigation. All nodes with valency of 2, i.e. with exactly one edge leading to it and one edge leading away from it, are being evaluated for removal. Nodes with valency 1 represent dead-ends and nodes with a valency greater than two depict junctions, both will not be removed since they are important to the road network.

In a first step all the considered nodes are sorted according to their deviation from a line connecting its two neighboring nodes. This way the nodes that distort the layout the most are considered first for removal. Then a check for the angle at the node considered for removal is made. The angle is measured between the edges leading from the considered node to the two neighboring nodes, and if that angle is within a certain range, that node is removed. This way sharp turns are retained, while nodes that represent only a small deviation from the direct line connecting its neighbors are removed.

It is important not to delete too much detail, as it may negatively impact the ability to navigate using the destination map. For this sake, roads further away from our destination are simplified more than roads in close proximity to it. Thus the details in little roads around our destination are retained, allowing for more guidance in the smaller areas of the map.

Additionally, despite not being mentioned in the paper by Kopf et al. [KAB+10], edges and nodes that do not connect to the destination node are removed during this step. Depending on the chosen area of interest, some parts of the visibility rings may not connect to the destination, but since all visibility rings are preserved throughout the algorithm, they may still be in the dataset. This makes this step valuable for a clean output.

3.5.7 Clipping

In the last step all parts of the data that are outside the bounding box entered by the user are removed. This prevents nodes and edges outside the bounding box from influencing the criteria evaluated during the road layout step. All nodes outside that bounding box and all edges with both their nodes being outside the box are deleted. Edges with one node inside and one outside, are split against the border and their outside node as well as the edge that is outside the box are deleted.

3.6 Data Layout

In the data layout step the remaining dataset is taken and multiple randomized moves are performed on the contained nodes (see Figure 3.6 for an overview of the used data structure). Every move changes the layout slightly and is evaluated by a cost function. If the cost of the newly changed dataset is lower than the cost of the previous data, the new set is used as next reference value, which is then changed by the next move, and so on.

This process aims to continuously make small improvements to the current result, until the improvement over 500 iterations of changing the layout is smaller than one percent. If that criterion is met for the first time, the cost function uses a different set of weights, thus changing which aspects of the layout influence this process the most. If it is met for a second time, the data layout step ends.

The changed layout is shrunk back into the original borders, if it grows outside of them. Additionally nodes which were on the boundary in the original dataset, are set to that border after every layout move. The cost function is easily changeable by adding new cost terms, which evaluate specific aspects of the data, or changing the weights associated with the current ones. Two newly proposed cost terms are described below.

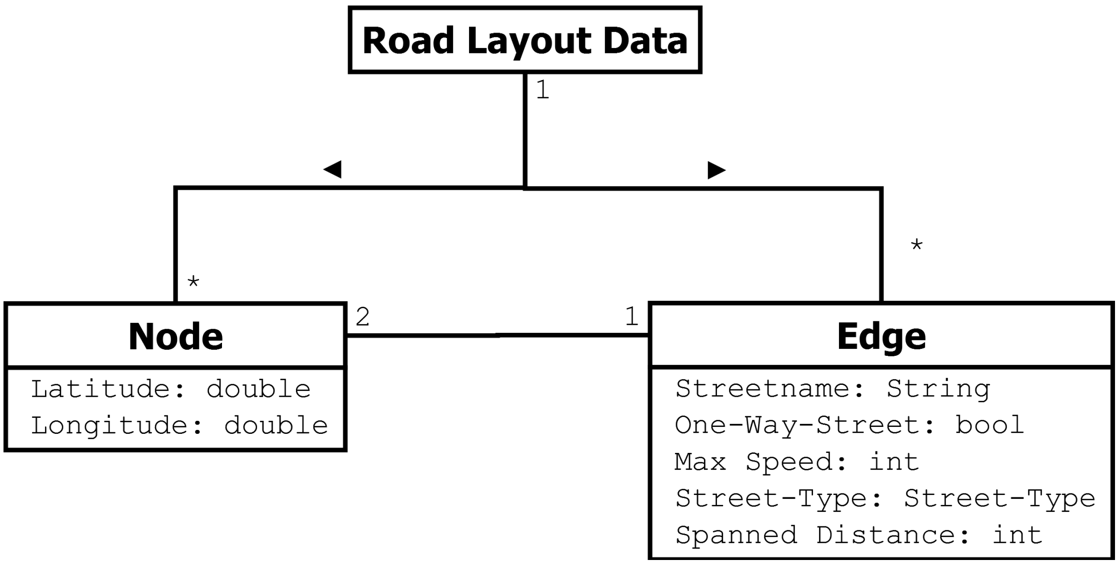


Figure 3.6: Slightly simplified schematic of the data structure used during the data layout steps. A road layout data object contains multiple edges and nodes. Edges contain all information associated with streets and reference exactly two nodes. Nodes simply contain the latitude and longitude coordinates of the given point. The diagram was made using the Dia diagram editor [Mac16].

3.6.1 Layout Moves

A single layout move consists of several, randomized steps. First off, a random node is selected from the dataset, through which an arbitrary line is laid. Then either the left or right side is chosen, and only nodes to the chosen side of that line are moved in the ensuing process. A scaling factor is then randomized, with values ranging from 0.95 to 1.05, in addition to a mode of scaling, either radial or orthogonal scaling. All nodes on the chosen side are then moved according to that scaling mode. Note that in case the scaling factor is randomized to be 1, a re-randomization is done. This is because a scaling factor of one does not change the layout and would result in unnecessary time spent on reevaluating the same dataset with the cost function.

The radial scaling creates a vector from the considered node to the selected one, multiplies that vector's length with the randomized scaling factor, and then places the node at the position reached by the new vector. In the orthogonal scaling the normal distance from the arbitrary line to the considered node is computed, and the node is placed at the position reached by changing the distance through multiplication with the scaling factor.

3.6.2 Cost Function

The cost function evaluates a number of heuristics for the data passed on to it, and sums them up to decide whether the data after the move is better suited to represent a useful destination map than the one before the move. In order to evaluate the usefulness of the given layout, multiple cost terms are called, which evaluate a specific aspect of that layout. The returned costs are multiplied by a set of weights and summed up before they are returned for comparison against the cost obtained with the data before the move was made. The following terms are used:

False Intersection

This term controls whether two edges intersect that should not intersect (i.e. two edges corresponding to streets that do not intersect in the input data, but intersect one another after a layout move). If such a false intersection occurs, no further computations are made since that specific layout automatically has a cost of infinity. This prevents any false intersections from appearing in our final result.

In addition to the original false intersection term [KAB^{+}10], the adapted term we propose in this work includes an exception for intentionally included false intersections, leaving two edges that falsely intersect in the unchanged dataset unpenalized if they intersect in the proposed layout. This enables datasets to work, even if they contain false intersections, e.g. if they contain a road on a bridge and a road running below it.

If no such false intersection occurs, this term inflicts penalties when two edges are located too close to one another, thus making the map hard to read. The reference for what is considered too close is tied to the output-format, as a distance smaller than 8 millimeters between two edges is penalized. The fact that the data layout depends on the output size is not ideal, because the same input data can result in different outcomes depending on the the output size.

The cost of Equation 3.1 is computed for every pair of edges in the dataset. As with all following cost terms, the summed up values are then multiplied by the weight associated with the term and current phase of the layout process.

$$false\ intersection\ cost = \begin{cases} \infty & \text{if false intersection} \\ 0 & \text{if known false intersection} \\ 0 & \text{if } actualDist \geq minDist \\ \left(\frac{minDist - actualDist}{minDist}\right)^2 & \text{else} \end{cases} \tag{3.1}$$

Minimal Length

The minimal length term is penalizing all road segments that are shorter than 25 millimeters in the output. This helps to emphasize the smaller roads, thereby assuring that the destination map is readable. In order to check for that requirement, a factor is computed to map the real-world size of edges to the size of the corresponding edges in the output map. Note that the distances used in Equation 3.2 are the distances on the map, and not the real world distances.

$$minimal\ length\ cost = \begin{cases} 0 & \text{if } actualDist \geq minDist \\ (minDist - actualDist)^2 & \text{else} \end{cases} \tag{3.2}$$

Relative Length

The relative length term aims to maintain the relationship of lengths between edges. This is to preserve the size-order of road-segments. Edges are compared to all the other edges with regards to the ratio of lengths between them. Edges go unpenalized when the ratio between their lengths is the same as the ratio in the simplified data, which has not been optimized with regards to its layout. In Equation 3.3 *origOne* and *origTwo* reference the length of two edges in the original dataset (before a layout move was made, but after the simplification steps were applied), while *newOne* and *newTwo* reference the corresponding edge's length in the newly proposed dataset.

$$\begin{aligned} relative\ length\ cost &= \left(\frac{r_{orig} - r_{new}}{r_{new}}\right)^2 \\ r_{orig} &= \begin{cases} \frac{origOne}{origTwo} & \text{if } origOne > origTwo \\ \frac{origTwo}{origOne} & \text{else} \end{cases} \\ r_{new} &= \begin{cases} \frac{newOne}{newTwo} & \text{if } origOne > origTwo \\ \frac{newTwo}{newTwo} & \text{else} \end{cases} \end{aligned} \tag{3.3}$$

Original Orientation

This term measures the angular deviation between the edge in the simplified data (i.e. the data after all simplification steps, but before the first data layout step) and the corresponding edge in the newly proposed data. The cost of each edge is equal to the squared deviation of the two edges in radians (see: Equation 3.4).

$$original\ orientation\ cost = \theta^2 \tag{3.4}$$

Relative Orientation

For this term all edges of the simplified data are analyzed with regards to edges that are either orthogonal, straight or parallel to them. All directly connected edges are analyzed considering their orthogonality (the edges share a node and form an angle of about 90 degrees) or straightness (the edges share a node and form an angle of about 180 degrees). Then for each edge directly visible to the analyzed edge all parallel edges are identified (the edges do not share a node and form an angle of about 180 degrees). All these pairs are tagged once at the initialization of the cost function and then evaluated for each edge of the newly proposed layout. The penalty is the squared angular deviation from the identified, ideal position, as can be seen in Equation 3.5.

$$rel\ orient\ cost = \begin{cases} \theta^2_{eOne,\,eTwo} & \text{if the edges are marked as parallel or straight} \\ (\theta_{eOne,\,eTwo} - \frac{\pi}{2})^2 & \text{if the edges are marked as orthogonal} \\ 0 & \text{else} \end{cases} \tag{3.5}$$

Relative Relative Length

This is the first of two experimental terms, created to try to improve the original algorithm [KAB+10]. This term is a replacement of the original relative length term and aims to allow smaller roads to grow more, without losing the relations between edges of the same or of a similar street-type. In order to achieve this, the length of the smaller present roads are no longer compared to the length of the larger roads of the layout, and vice versa. Note that this may reenforce one of the perceived problems of the original algorithm, namely "A lack of ability to understand the distances for each segment of the routes" [KAB+10].

This term is very similar to the relative length term, but applies a weight to each comparison of edges depending on the two street-types of the edges to be compared. This can be seen in Equation 3.6, where *relLenCost* stands for the normal relative length cost, as computed with Equation 3.3. For this comparison we first make a check which street-types are prevalent in the current dataset and build a list with the occurring street-types. If the edges' two street-types are not neighbors in that list, the edge's sizes are not compared at all. If the street-types are neighbors, the edge's ratio is multiplied with a small value and if they are of the same street-type, the ratio is multiplied by a larger value.

$$relative\ relative\ length\ cost = relLenCost \cdot weight_{(\text{streetTypeOne,streetTypeTwo})} \tag{3.6}$$

The weight depends on the street-types of the two edges and the amount of edges of the given types. 60 percent of the cost of a given edge is taken from edges of the same street-type as the current one, while the other 40 percent are taken from the neighboring street-types. Note that this is not circular, i.e. the highest street-type is not considered as a neighbor to the lowest street-type. We define the value currentTypeCount from Equation 3.7 as the number of edges with the same street-type as $t1$ and $t2$. The value of neighborTypesCount from Equation 3.7 is the amount of edges of both neighboring street-types. In case only one neighboring street-type exists, only the edges of the existing neighbor are counted.

$$weight_{(t1,t2)} = \begin{cases} \frac{0.6}{\text{currentTypeCount}} & \text{if } t1 = t2 \\ \frac{0.4}{\text{neighborTypesCount}} & \text{if } t1 \text{ and } t2 \text{ are neighbors} \\ 0 & \text{else} \end{cases} \qquad (3.7)$$

The classification as neighbors is determined by the OSM order [OSM16b]. Two street-types are considered neighbors not only if the two are directly next to each other (e.g. primary and secondary), but as well if no edges of street-types in between the two are present in the dataset (e.g. primary and tertiary, if no secondary streets are in the dataset).

This enables smaller edges to grow more and take up more space of the final output, without losing the information about the relative lengths of edges. This can be especially useful in cases where the small streets close to the destination are of complex nature. Additionally the largest roads of street networks are often not very complex, or even reduced to simple straight lines by the geometry simplification step. Such roads need not be the largest ones in the layout, because they are fairly easily traversable in a smaller scale.

Area Control

The area control term tries to force the layout to be spread over the whole output size, penalizing areas of the output that are under-populated with nodes. For this term five rectangles are created within the borders of the data and their boundary is subsequently increased until a desired number of nodes is located inside the given rectangle (see Figure 3.7 for an exemplary placement of the rectangles within a given bounding box).

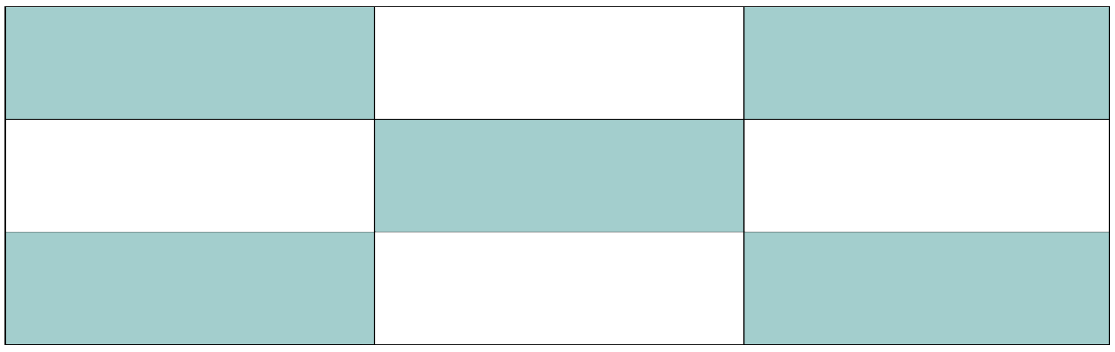


Figure 3.7: Example of how the rectangles are placed in the area control term. The blue tinted rectangles are the ones enlarged until the appropriate amount of nodes is placed inside them. Note that the height and width of the input bounding box are divided by 3 to get the correct placement of the lines forming the rectangles.

The rectangles' starting sizes as well as the increment of enlargement depends on the total area covered by the data. The amount of nodes necessary to stop increasing a rectangle correlates with the total number of nodes present in the data. The assessed penalty depends on how often a rectangle enlargement needed to happen for all rectangles to contain the necessary amount of nodes. The detailed procedure is depicted in Algorithm 3.1. As with the other cost terms, the resulting value, *usedSteps*, is weighted. Note that the area control term is the only one where the values are not summed up over the edges of the dataset.

Algorithm 3.1: Area Control Term

```
$Rectangles \leftarrow CreateRectangles()$
$NodeSet \leftarrow NodesInDataSet()$
$UsedSteps \leftarrow 0$
$RequiredNodes \leftarrow ComputeRequiredNodes(NodeSet.size())$
while $Rectangles$ not empty do
    $RectangleCount$
    for $i \leftarrow 0$ to $Rectangles.Size()$ do
        $RectangleCount.at(i) \leftarrow 0$
    end
    foreach $Node$ in $NodeSet$ do
        foreach $i \leftarrow 0$ to $Rectangles.Size()$ do
            if $Node$ inside $Rectangles.at(i)$ then
                $RectangleCount.at(i) \leftarrow RectangleCount.at(i) + 1$
            end
        end
    end
    foreach $i \leftarrow 0$ to $Rectangles.Size()$ do
        if $RectangleCount.at(i) \geq RequiredNodes$ then
            Remove $Rectangles.at(i)$
        else
            $Rectangles.at(i).IncreaseSize()$
            $UsedSteps \leftarrow UsedSteps + 1$
        end
    end
end
return $UsedSteps$
```

3.6.3 Cost Term Weights

The weights associated with the cost terms are changed during the optimization process. First a more aggressive set of weights is used, where changes to the general layout are less penalized than in the second stage, where the algorithm aims to reach a similar layout to

Cost Term	Weight 1	Weight 2
False Intersection	100	100
Minimal Length	100	100
Relative Length	1	1
Orientation	1000	5000
Relative Orientation	0	500
Area Control	500	300
Relative Relative Length	100	100

Table 3.1: The weights associated with the cost terms.

the original dataset.

The summed up cost is evaluated every 500th iteration. If the cost has changed less than 1% for the first time, the weights are changed. If this happens the second time, the layouting process is finished and the computed map is returned.

The difference between the two sets of weights boils down to the two orientation terms. In the first set of weights the relative orientation term is not at all considered, while the original orientation term has a significantly lower weight in the first iteration of the layout process. This allows the first round of computations to more freely change the layout, if it results in better solutions in the other areas. All weights can be seen in Table 3.1, note that the minimal length term has a weight of 100, instead of the weight of 1000 from the original paper [KAB$^+$10]. This is because in our algorithm the distances used to uphold the minimum length criterion are computed in millimeters instead of centimeters, thus the factor 10 is not missing, but moved to the minimal length computation.

3.7 Rendering

In order to render the data after the layout step, the Mapnik library is used to produce an image file of the given input data. The rendering specifications are loaded from a style-sheet, which is replaceable by the user if a different style of rendering is desired. The path to that style-sheet and the paths leading to the used fonts need to be declared in the settings file, as well as the resolution of the output. The final data is saved under a specific name and is then referenced as data-source in the style-sheet, which provides Mapnik with the to-be-rendered data as produced by the proposed algorithm.

CHAPTER 4

Implementation And Used Technologies

We implemented the Automatic Generation Of Destination Maps paper [KAB$^+$10] in C++ using Microsoft Visual Studio 2010 [Mic16]. The basic graphical user interface was made with QT [Com16] and the rendering is done with Mapnik [Pav16]. The used data is obtained via cURLpp [BL16] from the Open Street Map project [OSM16a] using Xapi. For the map data displayed in the user interface the google maps API has been used. The user interface can be examined in Figure 4.1.

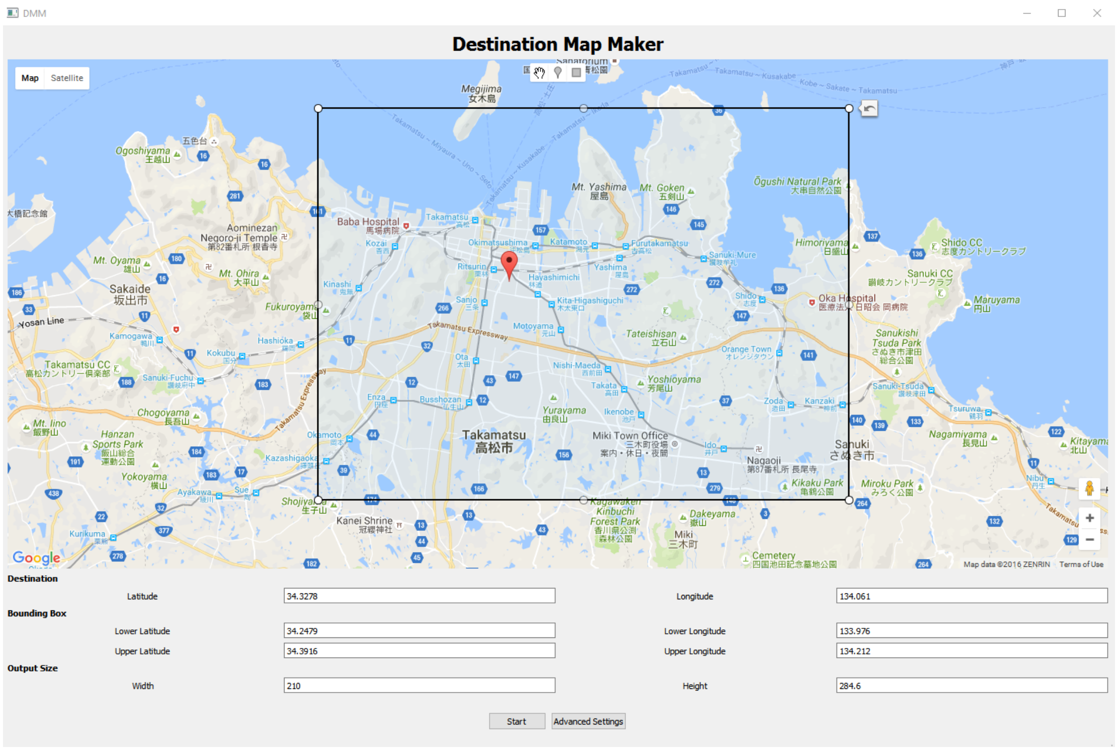
4.1 Open Street Map - OSM

The open street map (short: OSM) is an open source project providing street data, free-to-use, for anyone. The data of the OSM project is maintained by its users (similar to Wikipedia) who can easily edit or expand the current dataset. Areas with many participating users are well covered and changes in that area will often be reflected in the map data faster than with conventional mapping methods.

4.1.1 OSM-Data

The data saved in the OSM is organized in three different entities. A node, representing a single point in the database, which only stores information about its unique identification number and its world position in latitude and longitude values. An important attribute of a node is its valency, which equals the number of paths starting or ending at that node, e.g. the intersection node at a T-junction has a valency of 3, while intermediate nodes have a valency of 2.

Nodes are contained in ways, which describe the connection between points. Each way contains references to the nodes it connects, as well as additional information about the



Destination Map Maker

Destination

Latitude	34.3278	Longitude	134.061

Bounding Box

Lower Latitude	34.2479	Lower Longitude	133.976
Upper Latitude	34.3916	Upper Longitude	134.212

Output Size

Width	210	Height	284.6

Figure 4.1: The implemented user interface. The area of interest is set by drawing a rectangle with the corresponding tool displayed at the top of the map. Everything within that rectangle will be used as input data for the algorithm. The destination can be set with the marker tool. The values of the text input are automatically adjusted according to the rectangle and marker placed on the map, and vice versa. The algorithm is started by pressing the start button, after the area of interest and the destination have been set.

way itself, e.g. street-name or the street-type. The nodes referenced in a way are saved in a specific order, which is important when dealing with one-way roads.

Relations are the last type of entity. They describe some arbitrary relation between nodes and/or ways. These can be useful for marking bus-routes or all footpaths in a certain area. Relations however are not useful for our implementation, and are therefore not described in greater detail here.

All OSM entities can carry arbitrary additional information in the form of tags, which are key/value pairs. Tags can store any kind of information and, while there is a list of commonly agreed upon tags [OSM15a], any values can be set by users. This provides unlimited possibilities for saving information, but is highly un-standardized, which is an issue when parsing the OSM-data into another format.

Important tags used in this implementation are briefly introduced here. The "highway" tag is used to describe the type of street of the way entity (e.g. "highway=motorway", "highway=residential"). The "oneway" tag marks in which direction the way is traversable. Additionally the "maxspeed" tag contains the maximum speed allowed when traversing the edge. Default values for the "maxspeed" tag are applied based on the street-type of the edge (i.e. based on the value of the "highway" tag). The "junction=roundabout" key/value pair is used to identify nodes and edges belonging to roundabouts. Lastly the "name" tag is necessary to obtain the street name of the given way.

4.1.2 Data Access With Xapi

Xapi, the "OSM Extended API", is used in this implementation as method of data access. It is a read-only way to access the OSM database, with responses obtained following the standard protocol for OSM data.

Xapi requests take up to two parameters: the bounding box, limiting the area, and the tag-key/-value pair, restricting the output to results with the desired tag. The bounding box parameter consists of the positive and negative latitude/longitude values, thus creating a rectangular search area.

The tag-key/-value pairs can be applied to one of the main entities in the OSM data structure (i.e. node, way or relation). If ways or relations are queried, the result will contain the nodes referenced in the way or relation as well. The tag value restricts the output to only contain entities fitting the desired criterion (e.g. query for motorways with "highway = motorway", or for pubs with "amenity = pub"). A wildcard value can be used, which returns all entities that have the given tag independent of their value. Either the bounding box or the tag-key/-value pair need to be set for a successful query.

The biggest advantage in using Xapi is that the user needs not store the necessary information on the computer and is not responsible for updating or re-downloading it for sake of reflecting recent changes made to the map. This solution keeps the required storage light for users and provides up-to-date data, albeit with the necessity to download the information on the fly, which will result in slower execution times of the algorithm.

4.1.3 Issues

One of the issues with data obtained from the OSM project is that there are many different ways for naming. While some conventions are encouraged or discouraged one can not be sure about how closely the users creating that data are sticking to the encouraged norms. This variety of ways to capture the same information can result in problems when using the data as input, since the application may only work for certain naming conventions. As an example the OSM Wiki [OSM15b] shows the different ways of marking streets as one-way streets. There are three encouraged and five discouraged ways of defining a street as one-way street, which makes for a tedious process of getting the complete data, if the implementation is to cover all eventualities and multiple tags are to be interpreted.

Additionally the data may be underdeveloped in certain places where few voluntary helpers for the dataset are located. This potentially can make for certain applications working considerably worse in specific, less-covered locations.

4.2 Custom Data Format

The custom data format used in this work has a node class representing OSM nodes and an edge class showing a connection between two nodes (note that this is a different representation than the one with way entities in the OSM data; here an edge is showing just one connection from a node to another node, while an OSM way is containing all the nodes between two junctions). A lot of additional information obtained from the server is discarded, for it is not helpful in creating an easy-to-navigate road network. The resulting data is therefore only useful for navigational purposed, and not for inspecting details not pertaining to the road layout (e.g. the locations of hospitals will not be contained in the output even if hospitals were tagged in the input data). If so desired, this behavior can easily be changed by altering the input/output parsers and carrying that information through the optimization stages.

4.3 cURLpp

For access to the server providing the Xapi-service, the cURLpp [BL16] wrapper of libcURL [Ste16] is used. The server address is read from a settings file and used together with the Xapi request, as created by the algorithm, to make a valid URL to reach for the data. The raw obtained data is then written into a simple string for further usage.

4.4 TinyXML-2

In order to parse the obtained string and to print the data into XML files at given stages during the algorithm, the tinyXML-2 library is used [Tho16]. It allows for easily finding and reading interesting tags within the data to build the custom dataset from, as well as a good way of printing an XML version of that data.

4.5 Mapnik

Mapnik [Pav16] is used to render the output produced by the algorithm. For instructions on how to render the map, a style XML-file is read and the instructions in them are applied to the data at hand. That data can be provided in a multitude of ways, in this work it is provided as OSM-file (a feature that is enabled through a Mapnik plugin). The used values to render the data, described in a short fashion, are as follows:

- A *Map* is the root tag describing the to-be-rendered map as a whole. It sets the background-color as well as the type of projection used.
- A *Font* is used to register fonts that are later referenced when text is created. They determine the used typeface of the printed text.

- A *Style* contains multiple rules and simply has a name to be used as reference for the specified group of rules.
- A *Rule* specifies a group of instructions to be executed.
- A *Filter* is used in rules and makes all following statements within the given rule only applicable on a subset of the original data (e.g. the following tasks are only executed on motorways).
- A *LineSymbolizer* is used in rules. It draws lines of a given color and width, along the edges of the input data.
- A *TextSymbolizer* is used in rules and draws the names of the streets next to them. Contains values for which text to display, a reference to a font, the size of the text and parameters for where to place that text.
- A *Datasource* specifies where the given layer gets its input data from. Sets the type of input data (here to OSM data) as well as the filename.

4.6 Java OpenStreetMap Editor

The Java OpenStreetMap Editor (JOSM) [Edi16] is an editor, which operates on data in the OSM format, as produced by this algorithm. Figure 1.1 and all figures after Figure 3.2, including Figure 3.2 but excluding Figure 3.6, were created with JOSM.

CHAPTER 5

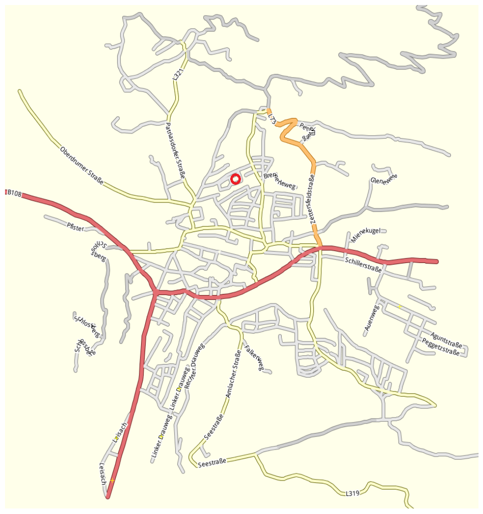
Results And Suggested Improvements

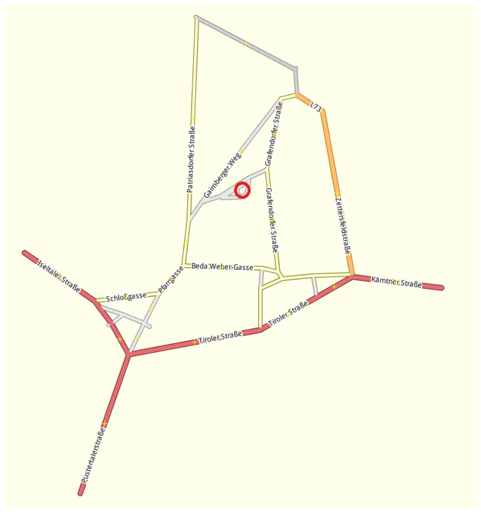
Some exemplary results can be seen in Figure 5.1. Issues with the algorithm can arise in various input regions, for specific constructs can sometimes negatively influence the outcome. While some of the difficult patterns have been dealt with in the algorithm, not all of them can be easily found and fixed, especially since a perfectly working solution is beyond the scope of this work. While efforts were made to improve the original algorithm [KAB$^+$10], issues may arise here, that were dealt with in the original work. This may be due to the sometimes unspecific explanations given in the paper.

The algorithm can be improved upon in cases where natural false intersections occur and in areas where lots of small details are needed for navigation. Further simplifications of complex intersections, e.g. at motorways, may improve the output, however such simplifications need to be made with care, since they might over-simplify streets in a different input area.

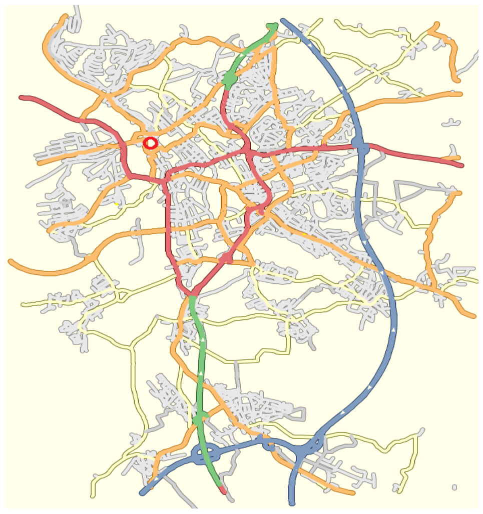
Figure 5.2 shows results obtained with the same data, but differing cost terms. The influence of the area control term seems to be negligible. Note how in the right dataset, the area control term has barely any impact at all. It generally suffers from forcing the data into the defined areas, without any consideration for the road network. The usage of the corner and center rectangles is only suitable for data that is structured the correct way. While a cost term that forces the data to use the whole output space seems like an improvement, a more sophisticated solution seems to be necessary to clearly improve the outcome. Another possibility would be to consider the area close to the destination node and specifically penalize the layout, if the area around it is too clustered.

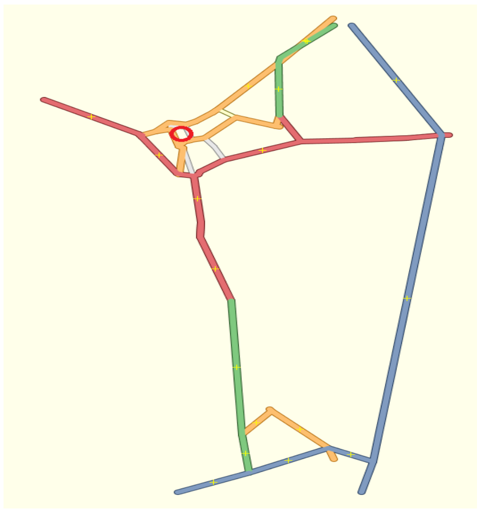
The relative relative length term, on the other hand, seems to fulfill its purpose and allows more growth for the smaller street-types, which is generally an improvement to

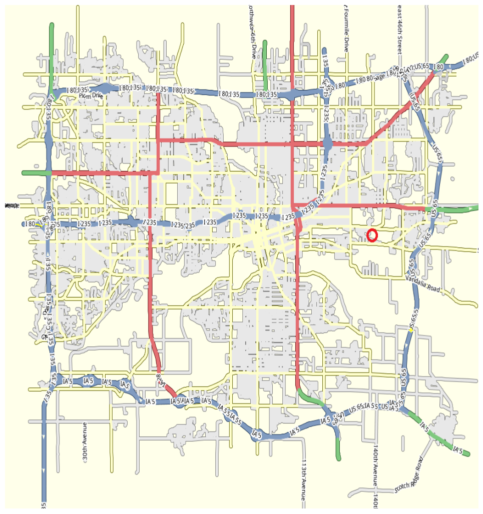

(a)

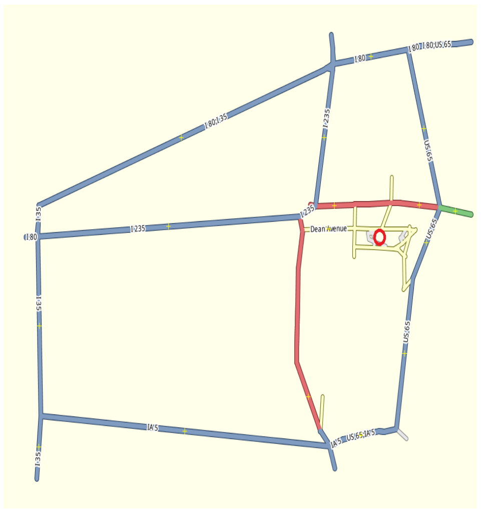


(b)


(c)


(d)


(e)



(f)

Figure 5.1: Results produced by the algorithm.

the layout. Note that the used formula to compute the term might be improved upon in future work.

In addition, an alternative to the relative orientation term might be necessary, since it only works well with many orthogonal angles in the dataset. This behavior is fine in areas where the roads are placed in a pattern similar to a chessboard. The term, however, has a negligible amount of influence on regions without such a pattern. A possible solution might be to increase the influence of the original orientation term, if only a small number of edge pairs for the relative orientation term (orthogonal, straight or parallel edges) were found.

5.1 Changes To The Original Algorithm

The changes to the original algorithm consist of the consideration of roundabouts, both during the selection of roads process 3.4.3 and during the simplification of roads process 3.5.1, the removal of streets that are only part of the dataset in order for travelers to change direction on motorways 3.5.5 and the newly proposed cost function terms, the relative relative length term 3.6.2 and the area control term 3.6.2.

5.2 Performance

The performance is highly correlating to the complexity, i.e. the amount of nodes and edges, in the input area. Note that this is especially true for all steps during the selection of roads part of the algorithm, since, after the selection is done, a big part of the complexity will be removed. Efforts to improve the runtime are therefore best utilized in the steps before the simplification begins.

The amount of nodes and edges alone is not the only factor in whether the current data takes long to compute or not. The structure of the data and the complexity remaining during the data layout stage, which is another part of the algorithm that needs to perform many costly computations, play a huge role in the achieved performance as well.

The time measurements shown in the Table below show that not only the amount of nodes and edges are important when considering runtime. Datasets with few nodes and edges may take long, if a lot of complexity makes it to the layout stage. The runtime measured consists of all parts of the algorithm, including the download of the data. The measured times do not include the proposed cost terms, but only the ones that are part of the paper by Kopf et al. [KAB$^+$10] and the time measurements have been rounded by truncation. The computer used for the measurements uses an Intel(R) Core(TM) i7-4710HQ CPU with 2.50GHz.

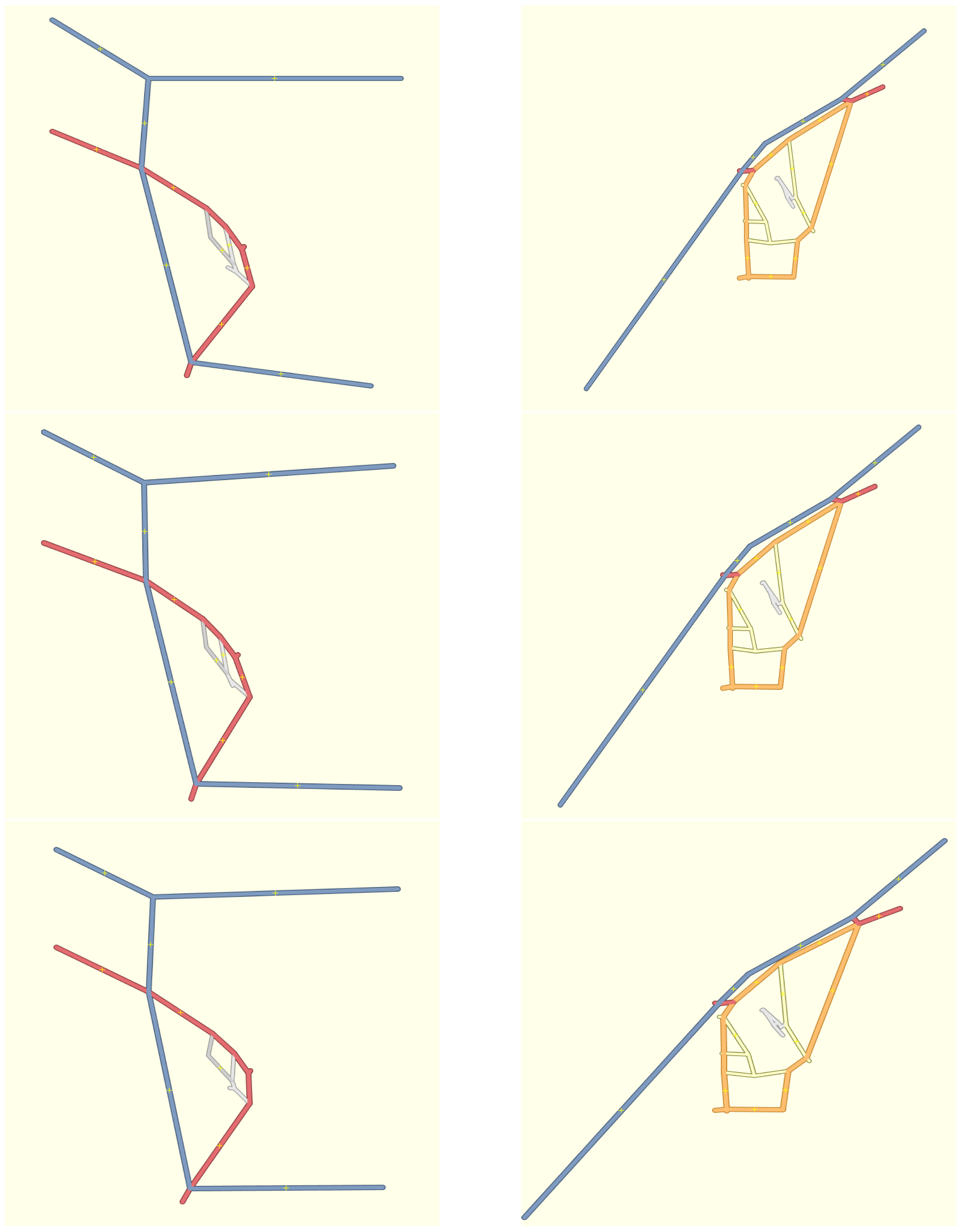


Figure 5.2: Results produced with different cost terms. The top row shows the cost terms as proposed by Kopf et al. [KAB$^+$10], the middle row shows images created by using the area control term, and the bottom row shows the relative relative length term, as replacement for the typical relative length term.

Runtime (min : sec)	Amount Of Nodes	Amount Of Edges
0:09	455	507
0:10	1341	1340
0:12	1926	1998
0:20	3536	3659
0:23	831	842
0:36	2573	2726
0:39	14301	15090
0:41	3919	4194
0:42	1781	1865
0:48	4117	4319
0:50	9250	9929
0:55	6705	6991
1:07	10409	10884
1:10	4388	4607
1:12	9121	10129
1:25	9520	9851
2:10	11163	11298
2:21	16250	17473
2:22	20731	22311
3:52	7072	7406
11:21	17616	18590

Table 5.1: The recorded runtime of the algorithm. We measured all steps taken during the creation of destination maps, including the download of the raw data. While, generally speaking, a smaller amount of nodes and edges should mean a shorter runtime, this is highly dependent on the specific input and no real conclusions can be drawn about the runtime from knowing the amount of nodes and edges.

5.3 Possible Improvements

The biggest way to improve upon the algorithm is to involve the user more in the process of creating the destination maps. A couple of suggestions follow to use the knowledge of the user to produce a better result. These steps should simply be offered in addition to the fully automated computing of destination maps, leaving the user the choice to invest more time in the process in order to improve and personalize the created output.

Another improvement could be made by redoing the user-interface to use the OSM data for the preview, since otherwise the data, as used in the algorithm, and the data, as used in the preview, will differ in some locations. This may leave the user with inexplicably missing, or differently named roads.

5.3.1 User Input For Road Selection

One of the ways of dealing with a point of criticism from the original algorithm, concerning missing favorite roads in areas the users have expertise in [KAB+10], is to involve them more in the process of selecting the roads to be taken and used in the layout. This additionally uses potentially existing knowledge of the user in that area, by allowing the user to review the selected roads.

At this stage the user sees all the roads that the program considers for the layout, before they are simplified whatsoever, with the non-taken roads still visible, but distinctly differentiable from the taken roads. The user may then change that information for various reasons. If, for example, the user knows some of the travelers are first at another location, which is not yet depicted, he may add it to the taken roads, with the algorithm connecting that new location to the already existing network.

The user should additionally be able to delete possible taken roads, if they are deemed unnecessary for that destination map. This process may enhance the choice of selected data with information the computer can not possess, but which is easy for the user to consider. E.g. a road may be undergoing reconstruction procedures, and may thus be completely blocked, which is why the user would want to remove said road. If the user chooses to remove one of the existing edges, a new shortest path algorithm needs be executed, which ignores the removed edge, in order to offer an alternative shortest route. The rest of the algorithm may then proceed as normal, without any further adjustments, with the steps for data simplification.

5.3.2 User Input for Simplification

Another method to improve the quality of the output is to present the user with the option to fix certain nodes of the layout, before the simplification process. These would be blocked from being deleted during the intermediate node removal step, similar to the way the destination node is retained, even though it has a valency of 2. A case where such an option would be helpful can be seen in Figure 5.1 (e) and (f), where the motorway edge in the top-left corner might be fixed by the user.

Such a feature would ideally be showing the data before and after the intermediate node removal side-by-side. The user might then fix nodes in the non-simplified data, with the cleaned up version updating after the input changes. This utilizes the user's preferences for retaining complexity where it is deemed necessary.

After the intermediate node removal, another input step might be ideal, where the user may remove nodes that were not removed by the automated removal. In this mode the user may simply be allowed to delete nodes and edges where necessary, reducing complexity where needed. Figure 5.3 shows an example where further simplification may improve navigability. The possibly unnecessary complexity stems from the serpentine roads leading up the mountains to our destination.

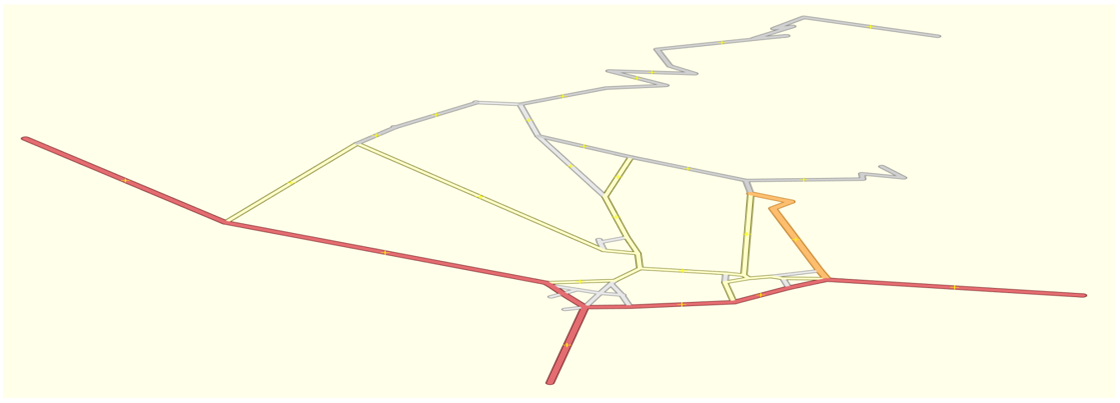


Figure 5.3: The simplified version of a road layout in the mountains. Note the complexity of the serpentine roads.

Bibliography

[AS01] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: Improving usability through generalization. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 241–249, New York, NY, USA, 2001. ACM.

[BL16] Jean-Philippe Barrette-LaPierre. curlpp main page. `http://www.curlpp.org/`, April 2016.

[BMWW14] Michael Birsak, Przemyslaw Musialski, Peter Wonka, and Michael Wimmer. Automatic generation of tourist brochures. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2014)*, 33(2):449–458, April 2014.

[Com16] The Qt Company. Qt. `https://www.qt.io/`, April 2016.

[Edi16] Java OpenStreetMap Editor. Josm main page. `https://josm.openstreetmap.de/`, June 2016.

[GASP08] Floraine Grabler, Maneesh Agrawala, Robert W. Sumner, and Mark Pauly. Automatic generation of tourist maps. *ACM Trans. Graph.*, 27(3):100:1–100:11, August 2008.

[Gol99] Reginald Golledge. *Wayfinding behavior : cognitive mapping and other spatial processes.* Johns Hopkins University Press, Baltimore, 1999.

[KAB$^+$10] Johannes Kopf, Maneesh Agrawala, David Bargeron, David Salesin, and Michael Cohen. Automatic generation of destination maps. *ACM Trans. Graph.*, 29(6):158:1–158:12, December 2010.

[KCJ$^+$10] Pushpak Karnick, David Cline, Stefan Jeschke, Anshuman Razdan, and Peter Wonka. Route visualization using detail lenses. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):235–247, March 2010.

[Mac16] Steffen Macke. Dia diagram editor. `http://dia-installer.de/`, October 2016.

[Mic16] Microsoft. Microsoft visual studio. `https://www.visualstudio.com/`, April 2016.

[OSM15a] OSM. Osm wiki - map features. http://wiki.openstreetmap.org/wiki/Map_Features, November 2015.

[OSM15b] OSM. Osm wiki - one way. http://wiki.openstreetmap.org/wiki/Key:oneway#Data_consumers, November 2015.

[OSM16a] OSM. Open street map main page. https://www.openstreetmap.org/, May 2016.

[OSM16b] OSM. Osm wiki - street types. http://wiki.openstreetmap.org/wiki/Key:highway, June 2016.

[Pav16] Artem Pavlenko. Mapnik. http://mapnik.org/, April 2016.

[Ste16] Daniel Stenberg. libcurl main page. http://libcurl.org/, April 2016.

[Tho16] Lee Thomason. tinyxml2 main page. http://www.grinninglizard.com/tinyxml2/, April 2016.

[Win02] Stephan Winter. Modeling costs of turns in route planning. *Geoinformatica*, 6(4):345–361, December 2002.