Institut für Theoretische Informatik
Lehrstuhl für Algorithmik I

Schematized Visualization of Shortest Paths in Road Networks

Diplomarbeit
von

Andreas Gemsa

an der Fakultät für Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Dr. Daniel Delling Dr. Martin Nöllenburg Dipl.-Inform. Thomas Pajor

Bearbeitungszeit: 15. Mai 2009 – 14. November 2009

Zusammenfassung

In dieser Arbeit untersuchen wir das Problem der Schematisierung von Routen in einem Straßennetzwerk. Ziel ist es für eine beliebige Route eine Ausgabe zu erzeugen, die eine schematisierte Fassung der Route sowie Dekorationen (z. B. Verkehrsschilder) aufweist. Bisher gibt es kaum Verfahren die es ermöglichen die Route mit verschiedenen Skalierungen der Teilbereiche darzustellen. Eine Route zwischen zwei Städten beginnt gewöhnlich mit einem kurzen Abschnitt in der Stadt beim Start. Es folgt ein längerer Abschnitt auf einer Autobahn oder einer Bundesstraße. Nach dem Verlassen der Autobahn oder Bundesstraße folgt in der Regel wieder ein kurzer Abschnitt, der innerhalb der Stadt des Zielortes liegt. Auf einer gewöhnlichen Übersichtskarte einer Route wird überall der gleiche Maßstab verwendet. Das ist allerdings für den Zweck einer Routenvisualisierung wenig von Vorteil, da das Erkennen von Details im Start- und Zielbereich deutlich erschwert ist. Wir verwenden verschiedene Skalierungen für verschiedene Bereiche. Das Ändern der Skalierungen führt zu neuen Problemen, die in dieser Diplomarbeit algorithmisch gelöst werden. Bisher gibt es nur einen Ansatz der diese Probleme versucht algorithmisch zu lösen. Dabei werden Verfahren eingesetzt die keinerlei Aussagen bezüglich Gütegarantien zulassen.

Die Ausgabe wird in mehreren Schritten erzeugt. Zunächst wird der Eingabe-Pfad in drei Teile zerlegt. In einen sehr kurzen Präfix-teil, einen sehr kurzen Suffix- und in den restlichen zentralen Pfad. Präfix- und Suffix-Pfad stellen dabei detailgetreu die Umgebung der Straßen im Bereich des Starts und des Ziels der Route dar. Der zentrale Pfad wird dann zunächst mit einem Algorithmus vereinfacht, damit die gesamte Größe der Instanz verringert wird. Dieser, nun vereinfachte, Pfad wird dann in monotone Teilpfade zerteilt. Diese monotonen Teilpfade werden mit einem von uns entwickelten Algorithmus schematisiert. Das heißt, wir beschränken die zulässigen Kantenrichtungen und verändern gegebenenfalls die Länge der Kanten. Dabei wird besonderer Wert darauf gelegt, gewisse Struktureigenschaften des Pfads zu erhalten. Das bedeutet, ein Knoten der in einem Teilpfad im Norden eines anderen Knotens des gleichen Teilpfads liegt darf nach der Schematisierung nicht im Süden dieses Knotens liegen. Sind alle Teilpfade schematisiert, werden sie zusammengefügt. Wir stellen ein Verfahren vor mit dem man Überschneidungen der Teilpfade beim Zusammenfügen vermeiden kann. Abschließend werden Straßenschilder plaziert, die dem Benutzer bei Kreuzungen anzeigen, welche Abfahrt zu nehmen ist.

Wir beschreiben den Schematisierungs-Algorithmus im Detail und zeigen seine Korrektheit. Außerdem erläutern wir die Laufzeit. Es werden dann weitere Techniken erläutert, die es ermöglichen, die Laufzeit des Algorithmus zu reduzieren.

Im vorletzten Kapitel beschreiben wir die Ergebnisse mehrerer Experimente mit unserem Verfahren. Dabei nutzen wir das Straßennetz Deutschlands als Grundlage. Wir besprechen

zunächst verschiedene messbare Eigenschaften unseres Verfahrens und analysieren dann die reale Laufzeit des Schematisierungsalgorithmus. Anschließend zeigen wir, wie sich das Verändern einiger Parameter auf die Ausgabe unseres Verfahrens auswirkt. Einige Beispiele der Ausgabe unseres Algorithmus werden dargestellt.

Contents

Acknowledgements

First of all, I want to thank Prof. Dr. Dorothea Wagner for giving me the opportunity to work on this topic. In no particular order, Dr. Daniel Delling, Dr. Martin Nöllenburg and Dipl. Inf. Thomas Pajor deserve special consideration. They all helped me immensely by offering support and good advice during the work on this thesis. In many long discussion they helped me find solutions to the various problem encountered.

I want to thank my parents. Without them I would not be able to have come this far. Also, I want to thank my siblings who both helped me throughout this thesis.

Finally, I want to thank the people who helped me with little things. For example, giving helpful suggestions or were just there for me when I needed a little distraction.

Dankeschön!

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt habe und nur die angegebenen Hilfsmittel und Quellen verwendet wurden.

Karlsruhe, den 13. November 2009

Chapter 1

Introduction

In recent years personal navigation system gained an increased popularity. There are several different manufacturers of hand held devices which guide users from their current location to any destination they choose. Furthermore, there are many popular internet route planning services, where it is possible to enter an origin and a destination between which the best path is computed. The result is usually a large overview map where the complete route, as well as other information is displayed. We call such a map a *route map* (or sometimes *route sketch*). Automatically generated route maps have often the disadvantage that they are cluttered. As an example, such an automatically generated route map is depicted in Figure 1.1 showing a route from an origin address in Karlsruhe to a destination in Frankfurt. Besides the map, additional textual route information is provided. The information given are, for example, "Turn right to merge onto A5 toward Frankfurt" or "Turn left at B39".

However, much of the information shown is unnecessary. For example names of towns as well as roads which are not close to the planned route or details of the route that do not require turning decisions by the driver could be omitted without losing important information. Since these maps are clippings of traditional road maps a single scale is used to display the whole route. This leads to the problem that small roads that are common at the origin and the destination of the route, are scaled so much that it becomes very hard to make them out. A route map provided by a human differs drastically from a route map generated by an internet service (see Figure for an example 1.2). It lacks much information which is included in the route map displayed by internet route planning services. A human omits unnecessary road. The length and orientation of roads are skewed. However, the information provided by such manually drawn of route sketches is sufficient to help travelling along the shown route. If we look at route sketches provided by humans they generally share the following features:

- Only roads which are part of the route or have a common intersection with a road of the path are shown.
- Roads are often depicted as straight lines. Much of the detailed geographic information is omitted.
- The road lengths are often changed. Longer roads are shortened, while shorter roads are lengthened relative to each other.
- The orientation of the roads is changed. The angles of the roads are only roughly similar to those in reality.

In this thesis we present an algorithmic approach for creating schematized route maps incorporating the previously shown features of manually drawn sketches.

There is a previous approach to automatically compute route sketches by Agrawala and Stolte [AS01]. Their system uses simulated annealing, which, however, has the drawback that no formal quality guarantees can be given.

On the other hand, Brandes and Pampel [BP09] showed that computing a schematized path that satisfies certain consistency constraints is an NP-hard problem.

Our Contribution. We provide and illustrate a technique to schematize any given route in a road network. The schematization process consists of several different steps which are explained in the following chapters.

The main ingredient in this process is a schematization algorithm An algorithm which takes as input a monotone path, a set of allowed directions and a minimum edge length. The algorithm computes a schematized version of the path, where the length of every edge is at least the provided minimum length and every edge is aligned to one of the allowed directions. Special consideration are in place to enable us to ensure that certain structural properties of the input path are maintained.

In order to apply this schematization algorithm to any given route, we describe a multi-step process that first splits the route into three parts: a prefix, a central and a suffix path. The central path itself is further split into a small number of monotone subpaths, each of which is schematized according to the above mentioned algorithm. These paths are then glued together and scaled so that the full route fits on a single sheet of paper. Finally decorations such as road signs or street names are added along the route in order to assist the driver.

So the result is an automatic process which can transform any given route into a easy to understand route map that fits on one single sheet of paper.

We have implemented our approach in C++. The average running time for any path is about 50ms. The results of several different schematized paths have been evaluated an can be found in the appendix.

Structure of the Thesis The Thesis is structured as follows.

Chapter 2. In this section we introduce some background about human cognition and how humans deal with their spatial surroundings. This gives a guideline on how to alter the depiction of a route so it is easier for humans to understand a route sketch.

Chapter 3. This section is devoted to previous work that is related to the topic we discuss in this thesis.

Chapter 4. In this section we introduce notations and terms which are important throughout the thesis.

Chapter 5. Here, we motivate all different stages of our route map design algorithm. The route which we want to schematize is splitted into different parts. The central path is schematized with the help of our schematization algorithm. This algorithm is the main part of our route map design algorithm. We motivate the schematization algorithm and give details why we chose our approach to solve the problem. Further, we give a description in pseudo code

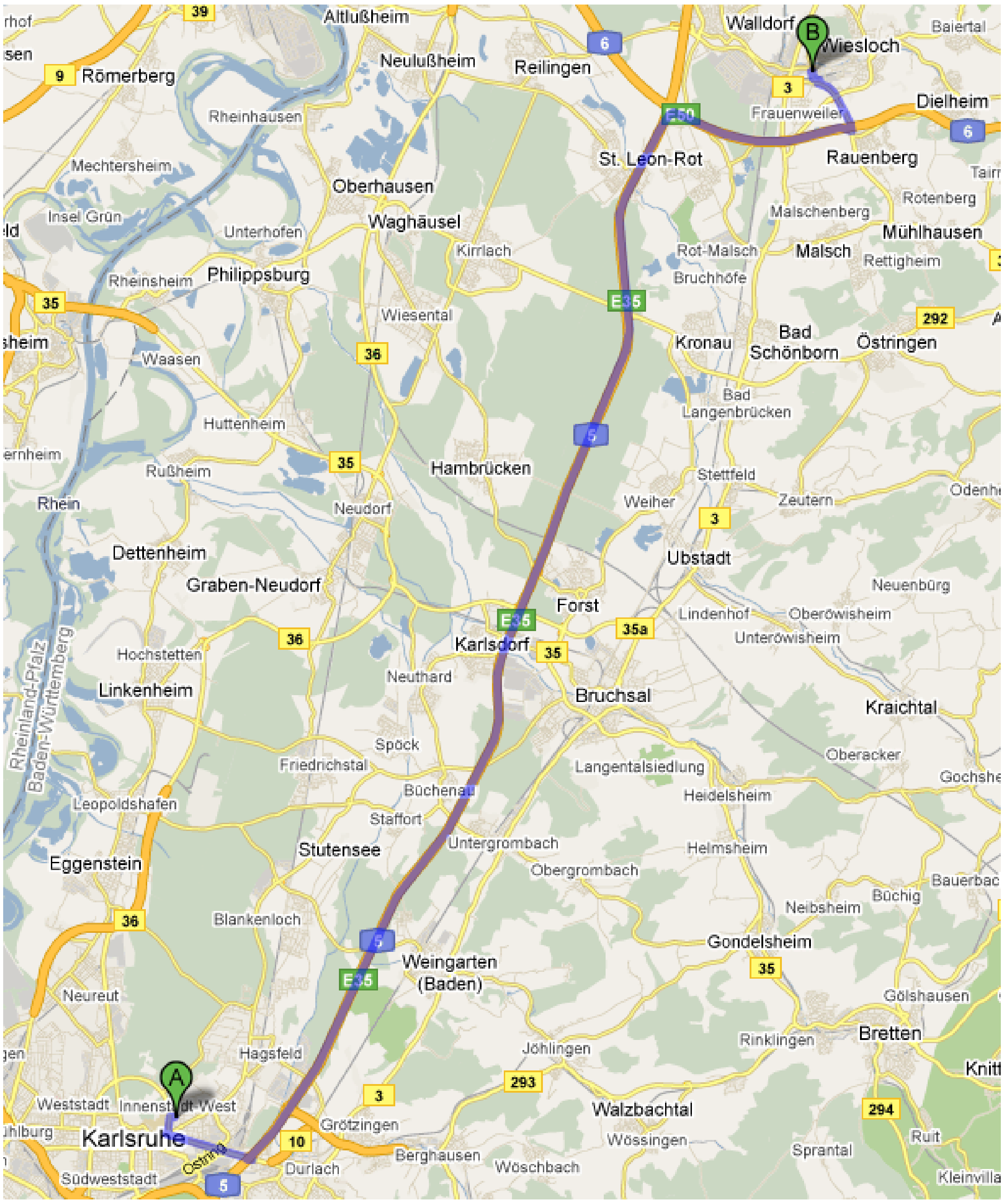


Figure 1.1: Generated Route Map by Google Maps (http://maps.google.de).

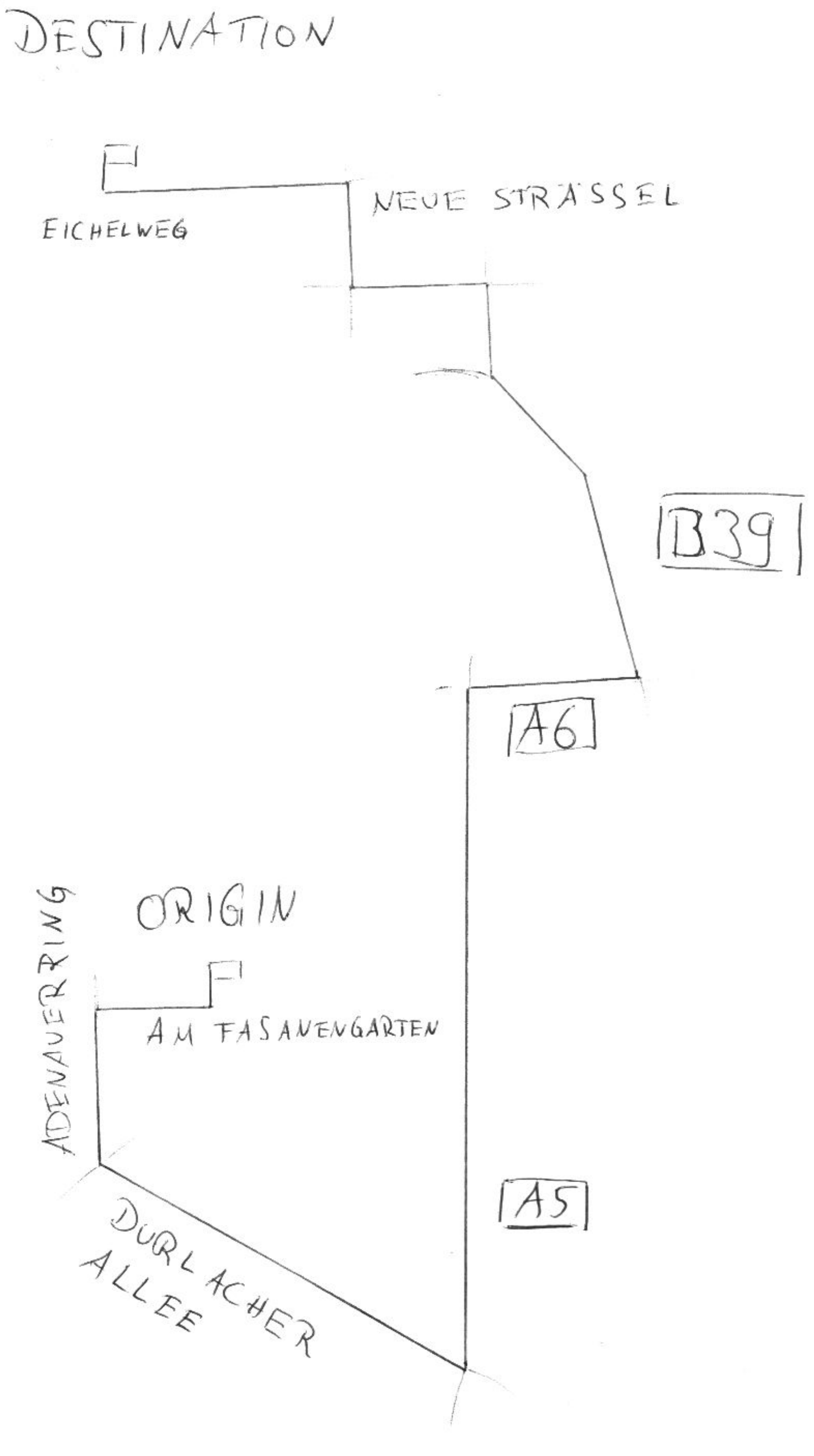


Figure 1.2: Route map drawn by a human.

and analyze its time complexity. A technique which can improve the running time of the algorithm is introduced. After discussing the schematization algorithm, we explain how it is used to schematized a given route and how the different parts of the route can be joined together. Finally, we scale and fit the route to the size of a sheet of paper and explain how to place street signs. These street signs get a position assigned which is chosen that no or at least only very few street signs overlap each other or a route segment. A heuristic which is able to achieve this is explained.

Chapter 6. In Chapter 6 we show results of several experiments that we conducted. There are three parts. The first part deals with measurable results, among which we also evaluate the running time of the algorithm.

The second part concerns aesthetic aspects. There are some parameters which can drastically change the shape of the schematized route. The same route is schematized with different parameters and the results are compared.

Finally, we show three different route maps our route map design algorithm produced. One route is a very short distance routes, one a medium length route and one a long route. The results are shown and compared the original geographic drawing of the route.

Chapter 7. We conclude this thesis in Chapter 7 and state several problems that came up during the work on this thesis and that motivate further research.

Chapter 2

Human Cognition

The problem of finding an optimal route between two points in a road network has been extensively studied and several results have been published. The fastest algorithms can compute an optimal route in road networks of whole continents within a few milliseconds [Dij59, DSSW09]. However, the question of visualizing the route on a single page in a clear way is a non-trivial task. If, for example, a person queries one of the numerous internet route planning services, he can normally see an overview of the route highlighted on a map. Additional textual information may be provided to help the user to drive along this route. The overview is generally not very helpful, besides conveying a certain "feeling" on the general layout of the route. The information displayed on these kind of overviews is often too much, thus, distracting the user from the essence of the route required for navigation. There are, for example, many streets and highways which the person following the route will never need to know about when driving along the route. Additionally, the streets are represented geographically, which means that every change in direction will be displayed on the overview. If the direction is only changed slightly it is not important for the user to know the precise geographic information.

In this thesis the general idea is to reduce as much information as possible while preserving necessary information which is essential for the user's orientation.

2.1 Human Spatial Cognition

Because our schematized route maps are targeted at humans, we have to discuss how humans process information. Here, we give a short introduction to several aspects of human cognition.

Self localisation. The first aspect we go into is self-localisation. Self-localisation is the ability of humans to be able to identify one's position in reality and find the corresponding position on, for example, a map. In order to find one's way when following a route one has to know about the current position to make the right decisions at the right time. There are different possibilities for humans to acquire knowledge about their current location. For example, if someone is in a town he is unfamiliar with, there are often large maps which depict the town as well as a marker which says "you are here" (Figure 2.1). This helps the person immediately. However, such maps cannot be placed everywhere across the town. If one is unsure about the current position humans search for features of their surrounding and compare them to the features presented in the map. Meiling et al. describe this behaviour

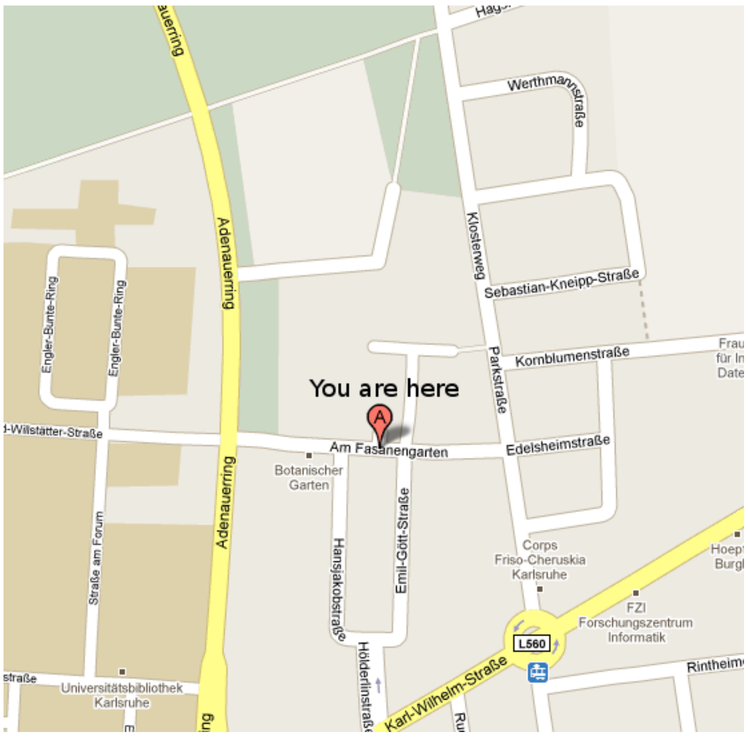


Figure 2.1: Example which show a person on is on a map (Generated with the help of Google Maps (http://maps.google.de/)

[MHBB06]. If, for example, one is at a T-intersection one will look for T-intersections in the map. Additionally, one will look for special properties (e. g., geographical) of the T-intersection to find the corresponding T-intersection on the map. But these *local cues* are not the only way to determine one's position. Humans can also take the *network structure* of the surrounding into account. This might be done to make sure that the correct position on the map has been identified. For example, a person sees on the map, a roundabout comes up, then this information can be used to validate or to find the position on the map.

Information Overload. The human brain has only limited capabilities. It is well known in the neurosciences that the human brains has only limited capabilities and that the short-term memory is very small [Mil56]. Much of the observed information is instantly forgotten. The human brain is severely limited in managing and processing information in this respect. Thus, it has to categorize information and decide what information to keep and what to forget. This implies that if we want to convey certain information (in our case routes in road networks) it is useful to remove redundant or irrelevant and distracting information. By doing that we enable humans not only to focus on the important parts (given that these important parts are still included), but we are also enabling them to acquire the provided information more quickly. The brain needs less resources to extract the important information, thus, the time needed for decisions is reduced.

Wayfinding Choremes. Klippel introduces the term choreme to describe "mental conceptualizations of primitive functional wayfinding and route direction elements"(pg. v, [Kli03]).

He performed a user experiment to find out how changes of directions are categorized. In one of the described experiments. The participants were asked to draw intersections where the route takes, e. g., a “half left” turn, or a “right” turn. The results are, as described by Klippel, that the participants used an underlying system to categorize the different turn directions into a 45° scheme unknowingly. As a result of this study Klippel constructed from the 8-direction model seven wayfinding choremes (Figure 2.2), called “sharp right”, ”right”, ”half right”, ”straight”, ”half left”, ”left” and ”sharp left”.

Mental Map. The mental map is a personal, internal representation of a visual depiction of a relationship between items. This can be, for example a diagram as described in [MELS95]. However, it is also used as a representation of a geographical map or a route map (see [Tve92, Bar02]). Unlike a real map, the mental map does not have a constant scale and consists of, for example, landmarks or regions. The spatial information is often not well modelled compared to Euclidean geometry. Turns are often remembered as right angles and curved lines are often straightened.

The mental map is how humans keep maps in their mind. It seems useful to adhere to the properties of mental maps in creating a schematized route. It will help the human brain to understand the depicted relationship better without destroying the ability to find the correct path.

This means we can remove curves in roads and simplify turning angles to emulate mental maps. At the same time our goal is to maintain certain aspects of the original path. For example, turning decisions have to be correct (i. e., if a left turn on the route map is depicted, the driver has to take a left turn also in reality, too).

2.2 Path Simplification and Schematization

A simple way to reduce unnecessary information in a drawing of a path is to “straighten” it so that little bumps are removed since this level of detail is not helpful for wayfinding. There are several methods to reduce points of a given polygonal path ([DP73, Far88, Ram72] or [BLR00]). Some of them are discussed in Chapter 3. An example of path simplification is shown in Figure 2.2. Although this seem like a very useful idea, applying it aggressively might yield unwanted results. It is also possible to use only certain angles for the edges of a path. A combination of both is depicted in Figure 2.3. Although the necessary information regarding the possible turning angles is included and correct from the drivers view at the intersection, the general directions (north, east, south, west) are lost. This may lead to certain routes where a town A which is north of another town B appears in the south of town B in the map. This produces confusion for the user and must be avoided as much as possible.

Another possible method to remove information is to consider intersections. At these points a decision has to take place. That means that at each intersection either the travelling direction changes (and the user has to take a turn) or the user crosses the intersection. Most intersections have in value three to five streets. For the driver it is not of utmost importance if the next turn he must take has a 95° angle or a 85° angle. In fact, Klippel’s wayfinding choremes indicate that only seven different directions are important as turning directions (see Figure 2.2). This of course applies to both left and right. This implies that if the visible angles are reduced to a 45° system, the map is simpler while still maintaining the most important information. However, it is conceivable that there may be paths that are be altered so drastically by this approach that the general directions (north, east, south, west)

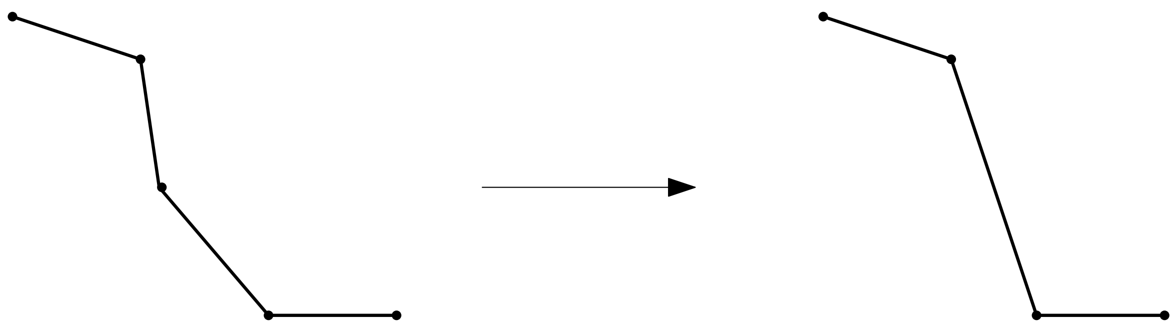


Figure 2.2: Example of path simplification

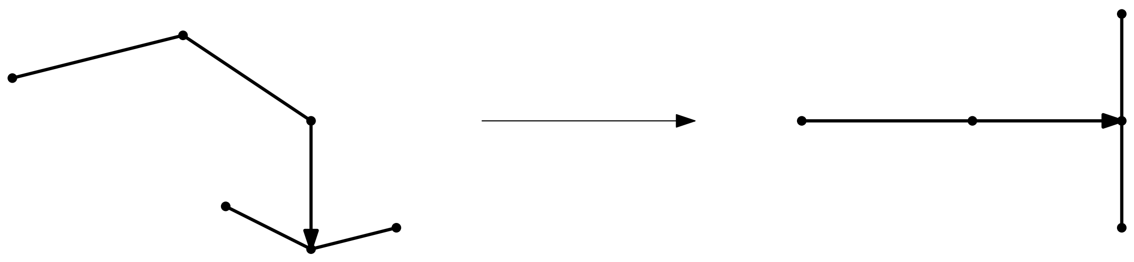
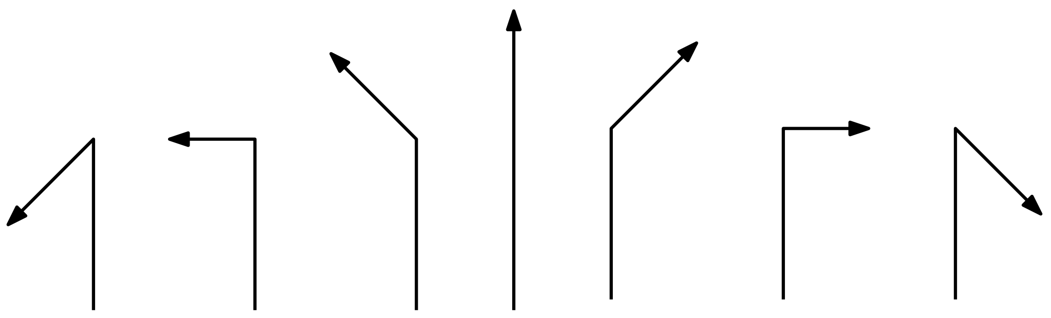


Figure 2.3: Example of path schematization which might change too much information

at some intersections may not be consistent with reality. If, however, the driver is instructed to drive straight through the intersection this information may be omitted. This presumes of course that the driver knows about this fact.

Another way of improving the readability of the map is to enhance it with other information that is useful for either orientation or wayfinding. At first glance this may seem counter intuitive because until now we only wanted to remove information. But consider a typical route, where a person wants to drive from his home town to a certain street in another town. The route will normally follow three simple steps. First, the person has to get out of his own town. He will need several turns at some intersections. The roads – or to be precise – the road segments he will use will generally be short. Second, if he has left his town he will most of the time come to a highway and travel on this highway until he approaches the destination. Third, he enter the destination town. There, he will again be travelling on short road segments and may need to take several turns at intersections. Consider such a route and how it looks if it has been straightened and the intersections have been simplified but the map scale is consistent for the whole route. The route will have at the beginning and at the end important information. However, it will be condensed to a relatively small part of the map. Unlike the part where the driver travels on the highway. This will make up for a rather large part of the map, but there will be very little important information for the driver. He will only have to change directions (change the highway or leave it) sometimes. This leads to the observation that the beginning and the end should be depicted larger relatively to the middle part of the route. So we enhance information by lengthening some line segments. Additionally, we use decorations, e. g., road signs (which may represent the already mentioned important turning angles).



(a) sharp left (b) left (c) half left (d) through (e) half right (f) right (g) sharp right

Figure 2.4: The seven wayfinding choremes

Chapter 3

Previous Work

Introduction. The Topic of simplification and schematization of paths or polygons has been a research subject for multiple scientists and is applied to multiple areas of real-life examples. In the following we present some selected research findings which were studied before preparing this thesis. Some of them present important mechanisms for our route map design algorithm. Others yielded inspiration for approaching certain problems which occurred while devising our algorithm.

Metro Maps. Metro Maps are a prime target for simplification. A part of the simplification process, as already indicated in the previous sections, is to enhance the readability of a path by removing unnecessary points [HMdN06, NW06, SR04]. An example for an algorithm which achieves just that is presented in [MG07]. Merrick and Gudmundsson present a technique which computes simplified polygonal paths in $O(|\mathcal{C}|^3 n)$ time, where n is the number of vertices in the original path and $\mathcal{C}$ is the set of restricted directions. The simplified path consists only of directions which are included in $\mathcal{C}$.

The general idea is to compute a *boundary path* and use this boundary path to construct the path simplification P. The boundary path is a sequence of straight lines which stab ϵ circles around the vertices of the path in the correct order. Merrick and Gudmundsson discuss further possibilities of their algorithm. They show how to avoid high bend angles and ensure a minimum edge length.

For metro maps this is a good approach. However, for our problem this technique has several disadvantages. The algorithm does not change the scale of the path which we determined as very important. Further, the orthogonal order is not necessarily maintained.

Schematizing Maps. Another approach is presented in [BLR00]. The authors use a technique called discrete curve evolution to reduce small and for the user unnecessary "bumps" to smooth out a given path or a given shape. Every point of a path or shape is categorized into three distinct classes. There are points which are *fixed*. Those points are not allowed to be removed because they represent a certain geographic feature. A removal would deteriorate the readability of the path. Further those points cannot be moved. The second class consists of *movable* points. As the fixed points, they cannot be removed, however their position can be changed. The last class consists of *removable* points. Obviously, they can be removed from the path. Then, the proposed algorithm calculates for every point of the path a value that represents its relevance. This is done with the help of a cost function, which depends on the length of the turn angle at that point and the length of the in- and outgoing edges. The points are sorted according to their respective value and it is determined which category they fit in. The algorithm then considers the point with the highest relevance in this sorted list. If the point is a movable point it may be moved (that depends on several conditions). If the point is a removable point it is removed. If the point is a fixed point it can neither be moved nor removed. Then the next point in the sorted list is considered and the evaluation continues again. If, however, there are only fix points inside the sorted list the algorithms aborts. This may also happen if another abort criteria is met. This approach can simplify and schematize paths as well as whole maps (e. g. subway maps).

This approach determines if points can be removed, moved or cannot be removed. This, however, is for our goals unsuitable. The scale for the path remains the same and this technique cannot guarantee that the orthogonal order is maintained.

Line Simplification. If only line simplification is desired, then one can use an idea presented in [Ney99]. The basic idea is that, given a polygonal chain P in the plane, a set of orientations $\mathcal{C}$, and a constant ϵ the algorithm computes a polygonal chain Q that consists of the minimum number of segments that have at most distance ϵ to P in the Frechet Metric (see [AG95, God91] or [Ney99, p. 10]). However, this approach is very restrictive and may not be suitable for many schematization algorithms. It confines the points to a certain area and does not remove unnecessary "little bumps" to help readability. This can only be achieved if this approach is combined with a path simplification technique (such as the mentioned discrete curve evolution).

Path Simplification. Another path simplification method different from the discrete curve evolution is the Douglas-Peucker algorithm. The original idea behind the Douglas-Peucker algorithm has been introduced in [DP73] in 1973 by Douglas and Peucker. Independently the algorithm has been suggested in 1972 by Ramer [Ram72] (thus, the algorithm is sometimes called Ramer-Douglas-Peucker algorithm). The algorithm's goal is to reduce the number of points in any given polygonal path. Simply put, is takes as an input the polygonal path and a parameter ϵ. It then draws a imaginary line between the starting and the end point of the path. The point p with the largest distance to the line is considered. If the distance is larger than ϵ p will remain a point of the path and the algorithm calls itself recursively with the starting point and p as parameters and p and the end point as parameters. The worst case running time is $O(n^2)$. The algorithm and improvements concerning the running time will be discussed in detail in a later section.

Hardness of Preserving the Orthogonal Order. In the paper [BP09] Brandes and Pampel discuss the hardness of preserving the orthogonal order when redrawing a given polygonal path. If a two-dimensional polygonal path $P = (v_1, \ldots, v_n)$ is given and we want to redraw this path then there is for every vertex in P a corresponding vertex in the new path P'. We denote the corresponding vertex to $v_i \in P$ as v_i'. The x-coordinate of a vertex is denoted with $x(v_i)$ and the y-coordinate likewise with $y(v_i)$. The orthogonal order is maintained if, for any pair of vertices $v_i, v_j \in P$ with $x(v_i) \leq x(v_j)$ the condition $x(v_i') \leq x(v_j')$ holds and for any pair of vertices $v_i, v_j \in P$ with $y(v_i) \leq y(v_j)$ the condition $y(v_i') \leq y(v_j')$.

They use `MONOTONE 3-SAT` which is known to be NP-hard (see [GJ79]) to prove two different statements. The first one is that if you want to redraw a path rectilinear (i. e., only the directions parallel to the x- or y-axis are used) the problem of deciding whether or not the orthogonal order can be maintained is NP-hard. The second one is that if you do not restrict the edge directions in any way but use only a uniform length for the edges the problem if deciding whether or not the orthogonal order can be maintained is NP-hard, too. Both proofs rely on a construction of special paths. They are constructed so that if you were able to draw the path rectilinearly or with uniform edge length in polynomial time `MONOTONE 3-SAT` could be solved in polynomial time.

Rendering Effective Route Maps. A similar problem as the one studied in this thesis is considered by Agrawala and Stolte [AS01]. Their general idea to solve the problem of displaying an easy-to-use map of a route is similar to our solution (which will be discussed in the following chapters). However, their approach is quite different to ours.

First, obviously, the travel route has to be computed. This is not part of their work but there are many known algorithms which solve the problem of finding a shortest path in a road network (see [Dij59, BD09, MSS+06] or [HSWW05]). Then, a shape simplification process is applied. The goal of this process is to remove segments of the road while still maintaining the overall shape of the route. The advantages of this approach is that this presents the user with a cleaner looking route as well as smaller memory footprint. It also yields faster processing times. Before removing segments of the path, for every vertex it is decided if it can be removed or not. This is done to ensure that there are no false intersections, no inconsistent path turns and no missing intersections. If a vertex would produce any conflict of the ones mentioned above the vertex is marked as unremovable.

The next step in the approach is to determine the road layout. Agrawala and Stole use simulated annealing (see [Fle95] for an introduction to this topic) to solve the problem. They compute an initial road layout by stretching all roads that are below a certain minimum length to exactly this length.

The perturb function which changes the road layout selects randomly a road r_i and changes its length by a random factor between 0.8 and 1.2. Further, the angle of the road is changed by a random angle between $+/-5$ degrees.

The score function which evaluates the quality for a given layout penalizes the following things: i), roads that have become smaller than the minimum road length, ii), two roads whose length orderings been swapped, iii), high difference of an angle of a road in reality and in the current road layout, iv), false or missing intersections are heavily penalized. Finally, with their approach it is possible that a destination which should appear to the north of the origin may appear in the south. Or the origin and the destination can appear to be closer too each other than they are in reality. To reduce this problem a vector between the origin and the destination is computed. A score is calculated that weighs the distance and the different angle between this vector in the original and the current road layout.

After the road layout has been determined, additional street labels and decorations are used.

Because simulated annealing is used the presented approach is not a deterministic process. This implies that it is impossible to guarantee certain quality properties of the resulting route map, most critically the orthogonal order of the route is not considered. Basically all objectives (road length, no false intersections, etc.) are target of the optimization, but it cannot be ensured that they are fully satisfied.

Chapter 4

Preliminaries

In this section present the notation and the introduce various terms needed for the description of the algorithm presented in Chapter 5.

Coordinates. We denote the x-coordinate of any given vertex v with $x(v)$. Accordingly, the y-coordinate of any given vertex v is denoted by $y(v)$.

Monotone path We say a path $P = (v_1, \dots, v_n)$ is x-monotone if for all v_i, $i \in 2, \dots, n$ $x(v_i) \leq x(v_{i+1})$ holds, or if for all v_i, $i \in \{2, \dots, n\}$ $x(v_i) \geq x(v_{i+1})$ holds.

Orthogonal Order. The concept of orthogonal order is explained in [BP09]. Preserving the orthogonal order aims to maintain the "mental map" a person has of a route.

Definition 4.0.1

Orthogonal order: Let $P = \{v_1, \dots, v_n\}$ and $P' = \{v'_1, \dots, v'_n\}$ be two polygonal paths with n nodes each. We say that P and P' have the same orthogonal order if for any pair $i, j \in \{1, \dots, n\}$ $x(v_i) \leq x(v_j)$ if and only if $x(v'_i) \leq x(v'_j)$ (and the same holds for the y-coordinates).

Basically it means that if we change the position of vertices of a path then a vertex v_1 that was to the left of a vertex v_2 must not be to the right in the redrawn path.

Input of the Algorithm. The input of the schematization algorithm is a polygonal path $P = (v_1, v_2, \dots, v_n)$ in the plane $\mathbb{R}^2$, where each vertex is a point $v_i = (x(v_i), y(v_i))$, $i = 1, \dots, n$ and a set $\mathcal{C}$ of allowed directions. The allowed directions are represented as angles (see Figure 4.1 as reference for the angles). We assume that $\{0, d_1, 90, d_2, 180, d_3, 270, d_4\} \subset \mathcal{C}$, where each d_i is a diagonal direction in the i-th quadrant. In our implementations we use regular angles that are either multiples of 30° or of 45° .

Note, for ease of description, we confine ourselves to x-monotone paths from left to right. However, the algorithm explained in the following section is capable of handling any monotone path. Further, we will assume that there are no two vertices of the path with the same y-coordinate. If this should happen, any vertex can be moved slightly so that this does not occur. Recall, we deal with real road networks. Hence, two nodes having the same y-coordinates is very unlikely.

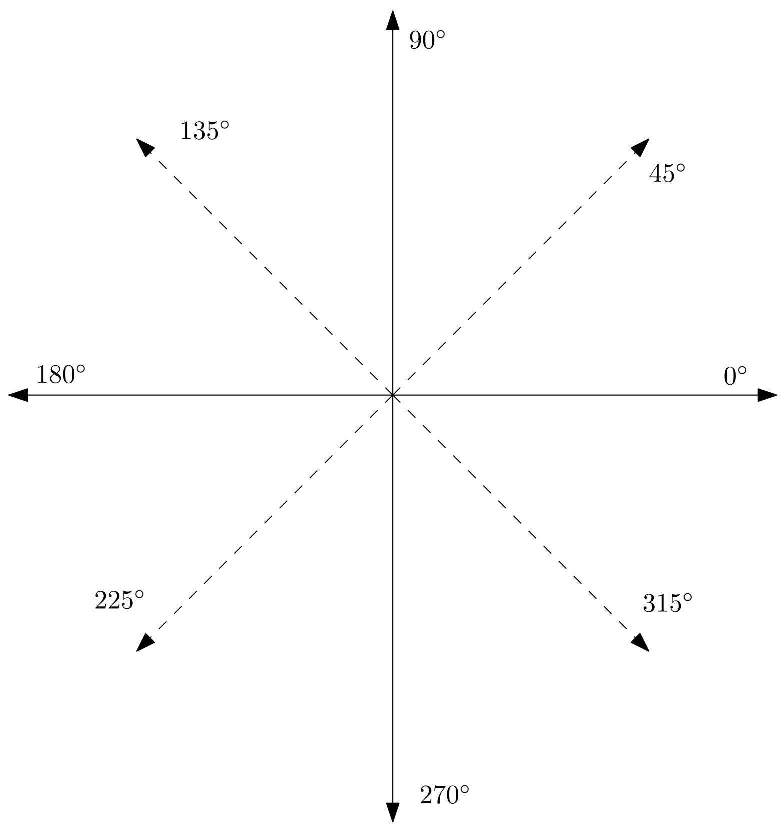


Figure 4.1: This figure illustrates how the angles are aligned.

Output of the Algorithm. The output of our schematization algorithm is a schematized polygonal path $Q = (q_1, q_2 \ldots, q_m)$ with $m \leq n$. Every vertex $q_i \in Q$ has a corresponding vertex $v_j \in P$. The path Q maintains the orthogonal order of the input path P.

Edges. Each pair of vertices (v_i, v_{i+1}) with $v_i, v_{i+1} \in P$ represents an edge of P.

Preferred direction. Each edge e has its *preferred* angle $\omega(e) \in \mathcal{C}$, from the set of allowed edge directions. The preferred angle (or direction) is the angle which is included in $\mathcal{C}$ and has the smallest difference to the original angle of the edge among all angles in $\mathcal{C}$.

Cost. The cost $c(e)$ for an edge e is 1, if it cannot be assigned its preferred angle. The cost is 0 if the preferred angle of an edge can be assigned to it.

Grouping of Edges. We group all edges into two classes. The edges whose preferred angle is $0°$ (see Figure 4.1) (i. e., all edges whose preferred direction is horizontal) are called **h-edges**. All other edges (i. e., all edges with a preferred angle that is not $0°$) are called **v-edges** since their direction has a non-empty vertical component.

Further Notation Regarding Edges We say an edge $e = (v_i, v_{i+1})$ is enclosed by two points v_j, v_k (without loss of generality $y(v_j) < y(v_k)$) if $y(v_j) \leq y(v_i), y(v_{i+1}) \leq y(v_k)$ and $y(v_i) \leq y(v_{i+1}) \leq y(v_k)$. Similarly, a vertex v is enclosed by v_j, v_k if $y(v_j) \leq y(v) \leq y(v_k)$.

An edge $e = (v_i, v_{i+1})$ *crosses* a vertex v if the endpoints v_i and v_{i+1} of e enclose v.

Horizontal Strips Given the position of all vertices v_i of P, we divided the plane into n horizontal strips $s_1, \dots, s_{n-1}$ from top to bottom by drawing a horizontal line through each point v_i. Let $y_l(s_j)$ and $y_u(s_j)$ be the lower and upper y-coordinate, respectively, that bound s_j. To each strip s_j the algorithm assigns a symbolic height $h(s_j) \in \{0, 1, -\}$ with the following meaning.

1. $h(s_j) = 0$: The height of the strip is zero. Thus, $y_u(s_j) = y_l(s_j)$ holds.
2. $h(s_j) = 1$: The height of the strip is 1. Thus, $y_u(s_j) > y_l(s_j)$ holds.
3. $h(s_j) = \ominus$: If the height of the strip is assigned $\ominus$, the height is (at least at this point) not important and can be either 0 or 1. However, this can change during the execution of the algorithm.

A horizontal strip s_i *affects* an edge $(v_j, v_{j+1}) \in P$ if $y_l(v_j) \leq y(s_i)$ and $y_u(s_i) \leq y(v_{j+1})$ (or vice versa if $y(v_{j+1}) \leq y(v_j)$), that is (v_j, v_{j+1}) crosses s_i.

We say a horizontal strip *s pushes an edge e vertical* if $h(s) = 1$ and s affects e. This means that e cannot be drawn horizontally.

A sequence of horizontal strips $\{s_i, \dots, s_j\}$ is identified by $s[i, j]$. The element with the index l inside $s[i, j]$ is denoted by $s_l[i, j]$.

Our schematization algorithm needs to store different strip height assignments. We user superscript to distinguish two different horizontal strip height assignments with a and a number. For example $s^1[1, n]$ and $s^2[1, n]$ are two different horizontal strip height assignments.

The algorithm will, in the end, assign each s_i a proper height instead of the symbolic height $h(s_i)$. We denote the true height of a strip s_i by $r(s_i)$.

Ordering path vertices by y-coordinates. In the following, the y-order of all vertices in P is important. Thus, we sort all vertices by decreasing y-coordinates and denote the i-th vertex in this order by u_i $(i = 1, \dots, n)$. This means that we have for a vertex $v \in P$ that $v = v_j = u_i$ if and only if v is the j-th vertex on the x-monotone path from left to right and the i-th vertex in the sorted sequence of vertices from top to bottom. The set of all u_i is denoted by U.

We define a bijection $\mu : \{1, \dots, n\} \mapsto \{1, \dots, n\}$ that maps the position i of a vertex v_i in P to the position of j of this vertex $v_i = u_j$ in the sorted top-to-bottom sequence. The inverse of the μ is denoted by μ^{-1}. Thus $v_i = u_{\mu(i)}$. Note that $y_u(s_i) = y(u_i) = y(v_{\mu^{-1}(i)})$.

Street Categories. Our algorithm deals with real road networks. In road networks are several different road types. A road inside a small town differs from a highway. The algorithm extracts the information from the underlying data set and assigns each road a certain category. Highways have the highest category. Small roads inside towns have the lowest. In total there are five different categories. Each road of the same category will later be drawn in the same color.

Chapter 5

Route Map Design Algorithm

In this section we introduce the main parts of our algorithm. It consists of several sequential parts which are illustrated in Figure 5.1. A rough overview is given in Section 5.1.

5.1 Basic Idea

Our approach can be divided into six different stages.

1. The input path is split into 3 parts. A prefix-, a central- and a suffix path are generated. The exact mechanisms are explained in Section 5.4

2. The Douglas-Peucker algorithm (in a slightly modified version) is applied to the central path (see Section 5.2).

3. This simplified path is splitted into monotone subpaths (explained in Section 5.5 and 5.6).

4. All subpaths are then schematized with our schematization algorithm (see Section 5.3).

5. Then, all schematized paths, the prefix and suffix paths are combined (see Section 5.7.

6. Finally, the combined paths are fitted on a DIN A4 paper and decorations are placed (see Section 5.8).

5.2 Path Simplification

The first step to reduce the complexity of the path is remove unnecessary vertices and to maintain only essential vertices of the path. There are some restrictions as to which points must not be removed. The details are explained later in this section. First, the general path simplification algorithm used is explained and later a refinement concerning the running time is discussed. Finally, we describe how to retain certain vertices.

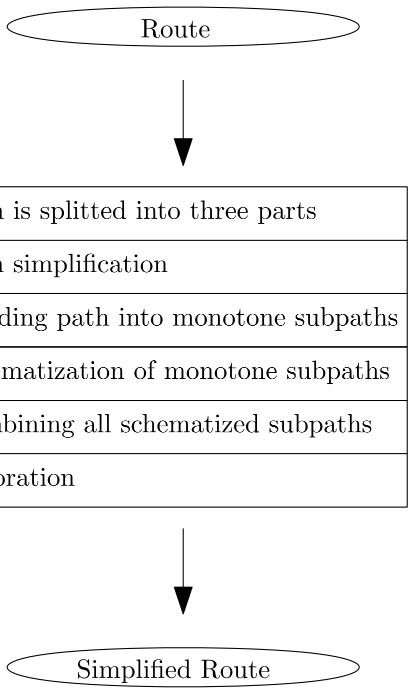

Route
Path is splitted into three parts
Path simplification
Dividing path into monotone subpaths
Schematization of monotone subpaths
Combining all schematized subpaths
Decoration
Simplified Route

Figure 5.1: Overview of the algorithm.

Douglas-Peucker Algorithm. The Douglas-Peucker algorithm (see Algorithm 1 for an illustration) has already been introduced in Chapter 3 and its basis is [DP73]. The input of the algorithm is a polygonal path P and a parameter ϵ is needed. This parameter is used to control the amount of simplification. Consider the path depicted in Figure 5.3. The algorithm is called by `DouglasPeucker(`P`, `ϵ`)`. The algorithm then creates a line l from `start` to `end` (in Figure 5.4 shown as dotted line and an ϵ strip around l). All vertices outside of the ϵ strip have a distance more than ϵ from l. In Figure 5.4 the algorithm determines that there are two vertices v_i and v_j which are outside of the ϵ strip. The algorithm chooses the vertex with the greater distance to l. In this case this is v_i. We call this vertex *splitting vertex*. Then, we recursively call `DouglasPeucker(`P_{v_1,v_s}`, `ϵ`)` and `DouglasPeucker(`P_{v_s,v_n}`, `ϵ`)`. If during the execution of a `DouglasPeucker` call there is no splitting vertex, the function returns the start and end vertex. Otherwise the function calls itself with the appropriate parameters. Finally, the algorithm returns a list of vertices which consists of the start and end vertices and all splitting vertices. The result for the example is depicted in Figure 5.6. Notice that all vertices but v_1, v_n and v_s have been removed. In the worst case scenario the algorithm is called $O(n)$ times recursively. During each iteration the calculation of the splitting vertices takes $O(n)$ time. Thus, the algorithms worst case running time is in $O(n^2)$. This is, for example, the case if ϵ is chosen to be 0. Then, the result is the original path and no vertices are omitted.

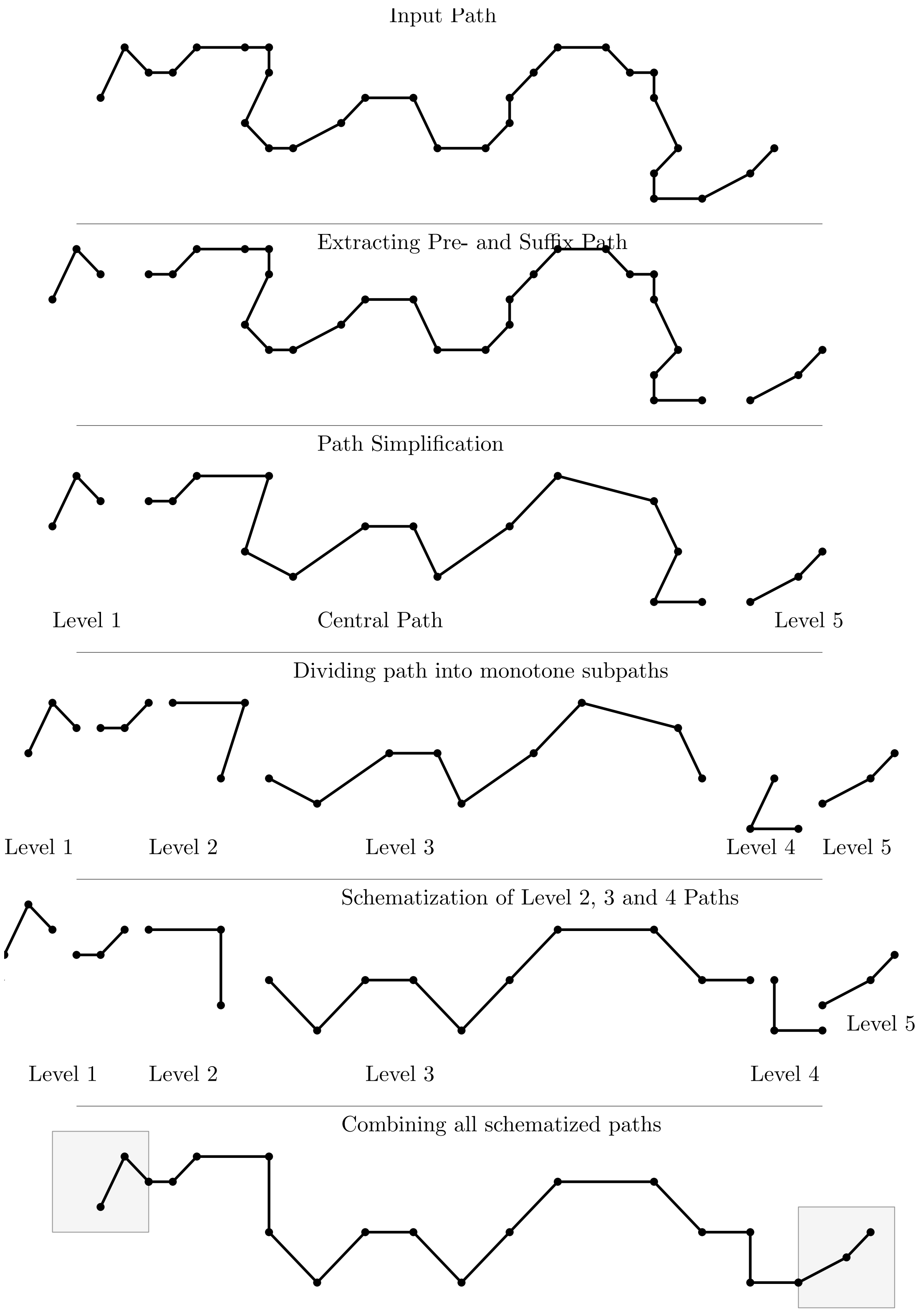

Input Path
Extracting Pre- and Suffix Path
Path Simplification
Level 1
Central Path
Level 5
Dividing path into monotone subpaths
Level 1
Level 2
Level 3
Level 4
Level 5
Schematization of Level 2, 3 and 4 Paths
Level 5
Level 1
Level 2
Level 3
Level 4
Combining all schematized paths

Figure 5.2: Each step of the route sketch algorithm is displayed.

Algorithm 1: DouglasPeucker

```
Input: embedded $x$-monotone path $P = (v_1, \dots, v_n)$ with coordinates $(x(v_i), y(v_i))$
        for $i = 1, \dots, n$, $\epsilon$
Output: Simplified path $Q = (q_1, \dots, q_m)$
1  $n = |P|$
2  max_dist = 0
3  max_dist_index = -1
4  Create line $l$ through $v_1$ and $v_n$
5  for $i \leftarrow 2$ to $n - 1$ do
6      dist$_i$ = Distance $l$ to $v_i$
7      if $dist_i > max\_dist$ then
8          max_dist_index = i
9          max_dist = dist$_i$
10 if $max\_dist > \epsilon$ then
11     Divide Path $P$
12     $P_1 = (v_1, \dots, v_{max\_dist\_index})$
13     $P_2 = (v_{max\_dist\_index}, \dots, v_n)$
14     sequence $se_1$ = DouglasPeucker($P_1$, $\epsilon$)
15     sequence $se_2$ = DouglasPeucker($P_2$, $\epsilon$)
16     return $se_1 + se_2$
17 else
18     return $(v_1, v_n)$
```

Improving Running time. As already mentioned, the Douglas-Peucker algorithm has a worst case running time of $O(n^2)$. In [HS92] Hershberger and Snoeyink propose an improvement to the Douglas-Peucker algorithm which change the worst case running time to $O(n \log n)$. The idea is to maintain a hull data structure (similar to that proposed by Dopkins et al. in [DGHS93]). This structural information is then exploited so that the next splitting vertex can be found in $O(\log n)$. It is also shown that this structural information can be maintained without deteriorating the running time while performing at most n splits. Thus, a worst case running time of $O(n \log n)$ is achieved.

Restrictions. In the introduction to this section it was hinted that there should be some restrictions as to which points are removed. The most obvious points which must never be removed are the ones at intersections where the driver has to take a turn. We call these vertices *decision points*. Further, we do not want to remove vertices where a street category change occurs. For example, if the driver changes from a main road onto a highway. Even though the road may not change its direction drastically, it is helpful to know that there is a change in the road category. vertices which are at a highway junction are important, too, if the driver needs to drive onto another highway. A special case are roundabouts. To the algorithm they appear as a sequence of multiple intersections in a very short distance. However, to the driver it is obvious that he drives onto a roundabout and the driver needs to know which exit to take. It is not important to illustrate every possible intersection but it is sufficient to keep only one vertex of the roundabout. All other vertices of that roundabout can be removed. The information which exit to take is displayed with the help of a road sign.

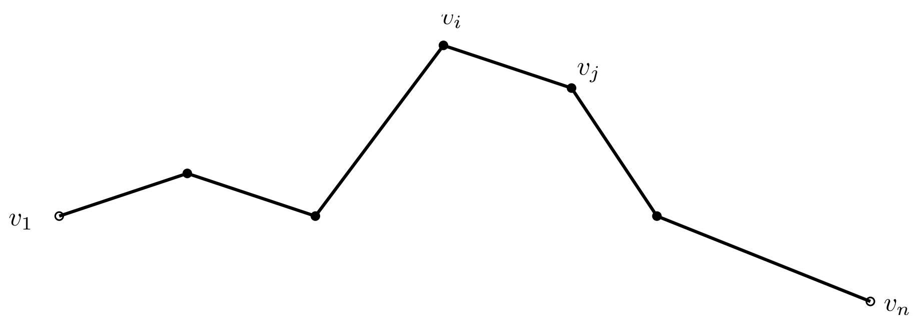


Figure 5.3: Douglas-Peucker algorithm: Step 1. The original Path.

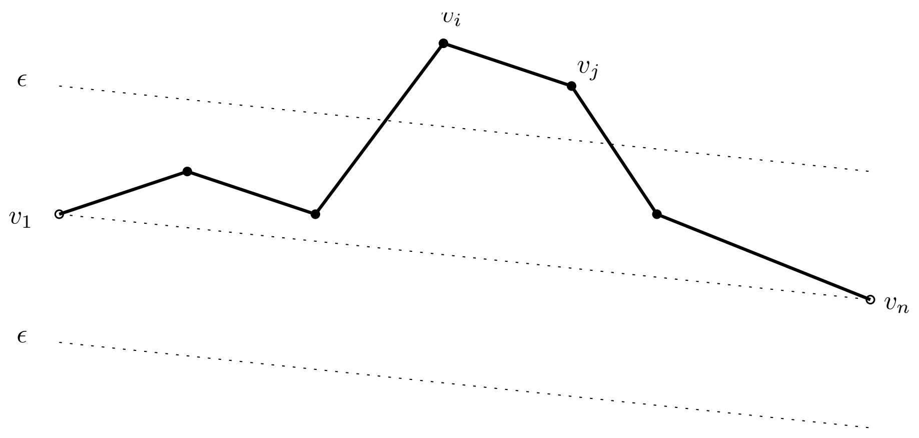


Figure 5.4: Douglas-Peucker algorithm: Step 2. Line Through v_1 and v_2 and the ϵ strips are drawn. The algorithm has identified the splitting vertex v_i and it is called recursively with two subpaths. One subpath begins at v_1 and ends at v_i. The other begins at v_i and ends at v_n.

If this were the only important points to keep, the resulting path might have inconsistent turn directions. An example when something like this can happen is shown in Figure 5.7(a). Although the driver has to take a left turn, the simplification produces a path which implies that at this intersection the driver has to take a right turn. This is undesirable and can be avoided. The technique used to identify points which must not be removed is based on ideas proposed in [AS01]. Assume that we have already determined all intersections. In Figure 5.7(a) we can see a path with intersections marked by a small dot. All vertices which are not intersections are marked with an x. We now want to determine the vertices which are marked with an x and can be removed without causing inconsistent turn directions. We now consider the vertices between two consecutive intersections v_i and v_j. We define a line l_1 through v_{i-1} and v_i. The line l_1 divides the plane into two half planes. We check if v_{i+1} and v_j are in the same half plane. If this is the case the turn directions are consistent. If this is not the case we mark v_{i+1} as unremovable. Then, a line l_2 through v_j and v_{j+1} is defined. The line l_2 divides the plane into two half planes. We check whether v_{j-1} and v_i are in the same half plane. If this is note the case, we mark v_{j-1} as unremovable. Otherwise, no adjustments are necessary. The result is depicted in Figure 5.7(b). The same procedure is done from the

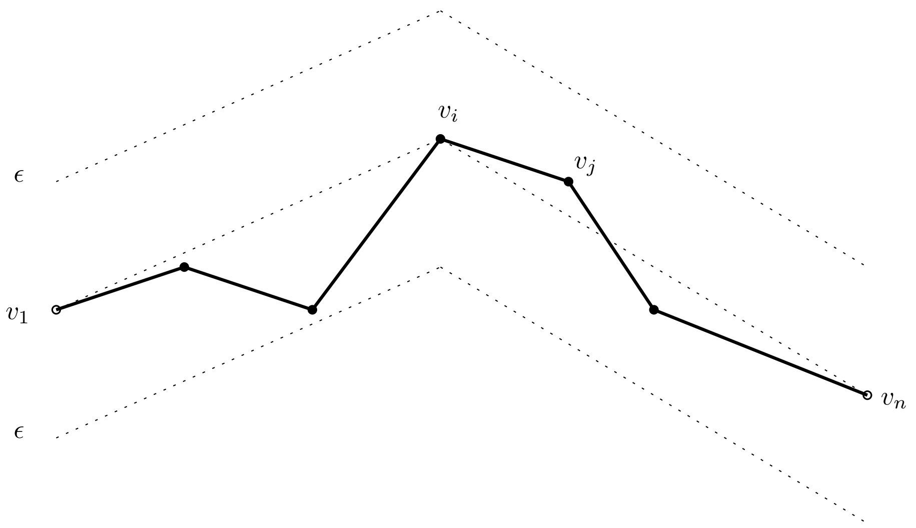


Figure 5.5: Douglas-Peucker algorithm: Step 3. The algorithm has been called again with two subpaths. Again, the ϵ strips are drawn. There are no splitting nodes left.

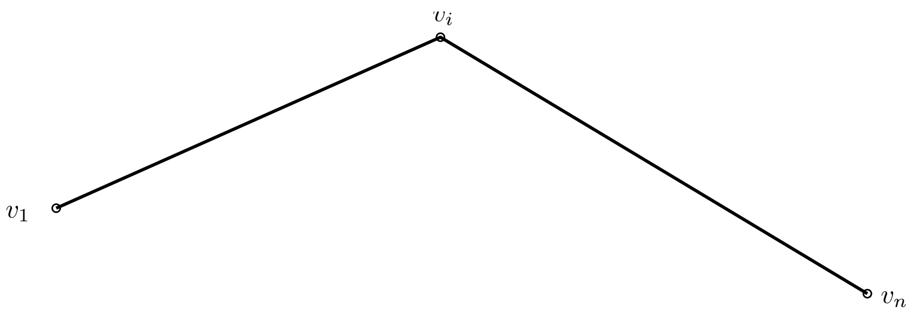


Figure 5.6: Douglas-Peucker algorithm: Step 4. The resulting path after the Douglas-Peucker algorithm has been applied. Note that all nodes except the first, the last and the splitting vertex have been removed.

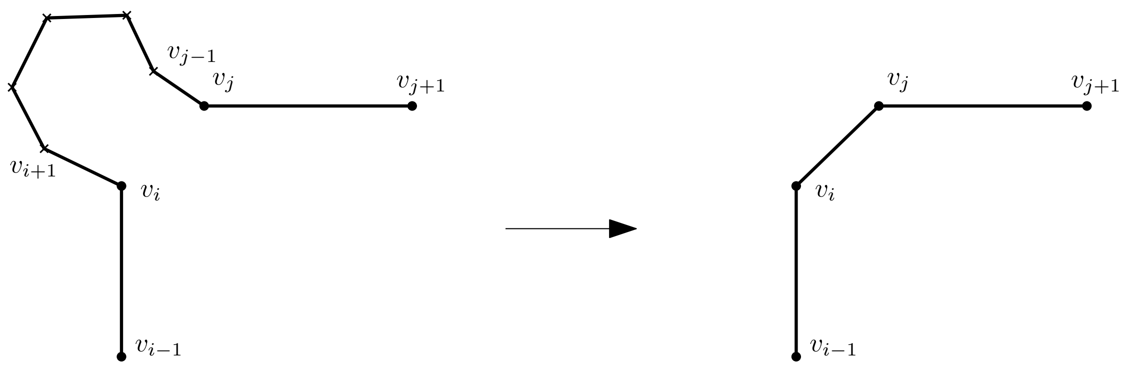
opposite direction.

5.3 Schematization Algorithm

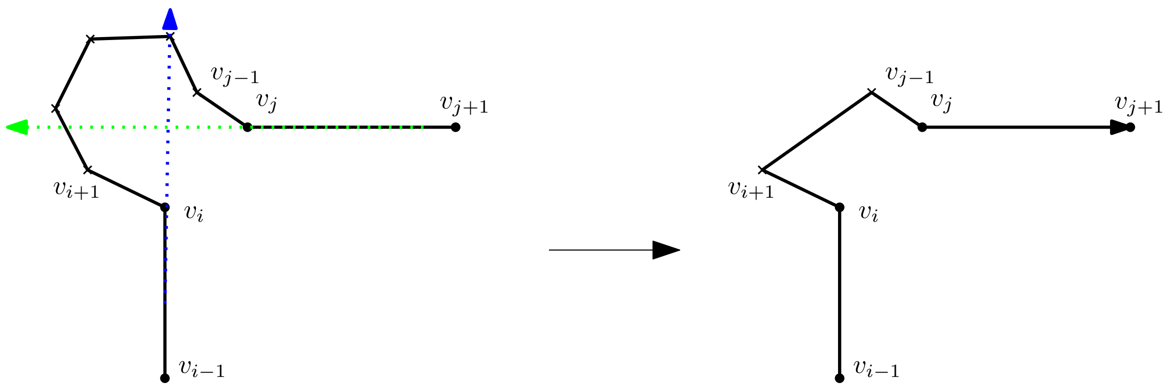
In the following we describe our schematization algorithm. First, we describe what our algorithm aims to achieve. We motivate briefly why the problem can become quite complex and naive approach may fail to address certain cases. Finally, we provide the schematization algorithm in pseudo code and explain the details. The algorithm devised is able to solve the problem given in the following.

Problem 5.3.1

Given a path $P = (v_1, v_2, \ldots, v_n)$ in the plane $\mathbb{R}^2$. Find a schematized path $Q = (q_1, q_2, \ldots, q_n)$ such that the orthogonal order is maintained and the number of edges which cannot be assigned their preferred direction is minimal.



(a) Simplification maintaining consistent turn directions



(b) Naive simplification

Figure 5.7: On the left side of the upper figure there is a path depicted which is to be simplified. The vertices marked with a black circle represent decision points. Vertices marked with an x depict vertices that might be removed.

5.3.1 Objective

Maintaining the orthogonal order. The goal of our schematization algorithm is to preserve the orthogonal order (see Definition 4.0.1 on page 19) of all vertices of the path. The orthogonal order of a path $P = \{v_1, \dots, v_n\}$ is maintained in a new path $Q = \{q_1, \dots, q_n\}$ if the following holds. Let $v_i, v_j \in P$ with $y(v_i) \leq y(v_j)$ then for the nodes $q_i, q_j \in Q$ the expression $y(q_i) \leq y(q_j)$ holds true. The same condition must hold for the x-coordinates. If one of the conditions for any node pairs does not hold, the redrawing of P to Q has violated the orthogonal order. Loosely spoken, this means if one considers two vertices and places one vertex in the point of origin, the other vertex is in one of the four quadrants (see Figure 5.8). If, in the redrawn path Q, the corresponding vertex is in the same quadrant the orthogonal order holds for these two vertices.

Brandes and Pampel [BP09] proved that the problem of drawing a path rectilinearly while maintaining the orthogonal order is NP hard. The same holds true if we allow all edge directions but restrict ourselves to uniform edge lengths.

The schematization algorithm we present is suited for monotone paths. We exploit certain properties of monotone paths to ensure minimal cost while maintaining the orthogonal order. To ease the description, we confine ourselves to an x-monotone path P with the start vertex v_1 on the left and the end vertex v_n on the right. The algorithm takes as input a monotone

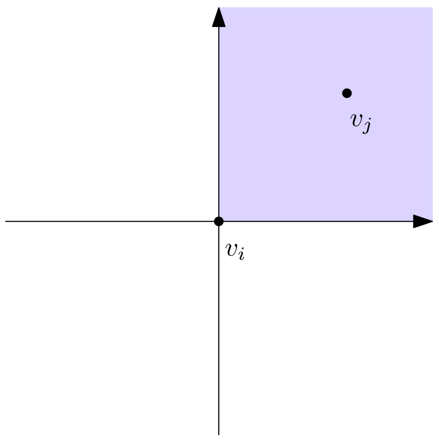


Figure 5.8: Two vertices v_i and v_j are shown. The orthogonal order is maintained if v_j is in the upper right, gray area (according to v_i).

path and a set of angles. This set includes all allowed angles (or directions) an edge can be aligned to. Further, We assume that $\{0, d_1, 90, d_2, 180, d_3, 270, d_4\} \subset \mathcal{C}$ (see Figure 4.1). This basically means that in addition to the horizontal direction, there is at least one direction to the upper right and lower right quadrant.

Minimizing Path length. We want to ensure that the total path length is as small as possible.

Combined. We consider two different optimization criteria:

- Minimizing the number of edges that are not assigned their preferred direction.
- Minimizing the total path length.

If only the path length is to be minimized, the solution is simple. A feasible length-minimal path would consist of horizontal edges only. This is, of course, nothing but a straight line from left to right. This is undesirable and in practice violates the "mental map" of the user (see [MELS95] for an overview) because it alters the path very drastically. Thus, the main goal is to minimize the number of edges that cannot be assigned their preferred angles without violating the orthogonal order. An example when this can happen is depicted in Figure 5.10. On vertex is connected by both a v- and an h-edge. In this special case not both of them can be assigned their preferred direction without violating the orthogonal order.

Lemma 5.3.1

For any given x-monotone path P setting the y-coordinates of all vertices to the same value does not violate the orthogonal order.

Proof: Assume that there are two vertices $v_1, v_2 \in P$ and their corresponding vertices v'_1, v'_2 with the same y-coordinate. The orthogonal order would be violated if $x(v_1) \leq x(v_2)$ and $x(v'_1) > x(v'_2)$ (or the other way around). Because the y-coordinates are the same, a violation of the orthogonal order cannot happen here. However, we did not change the x-coordinate of any vertex. This means that the orthogonal order is preserved ■

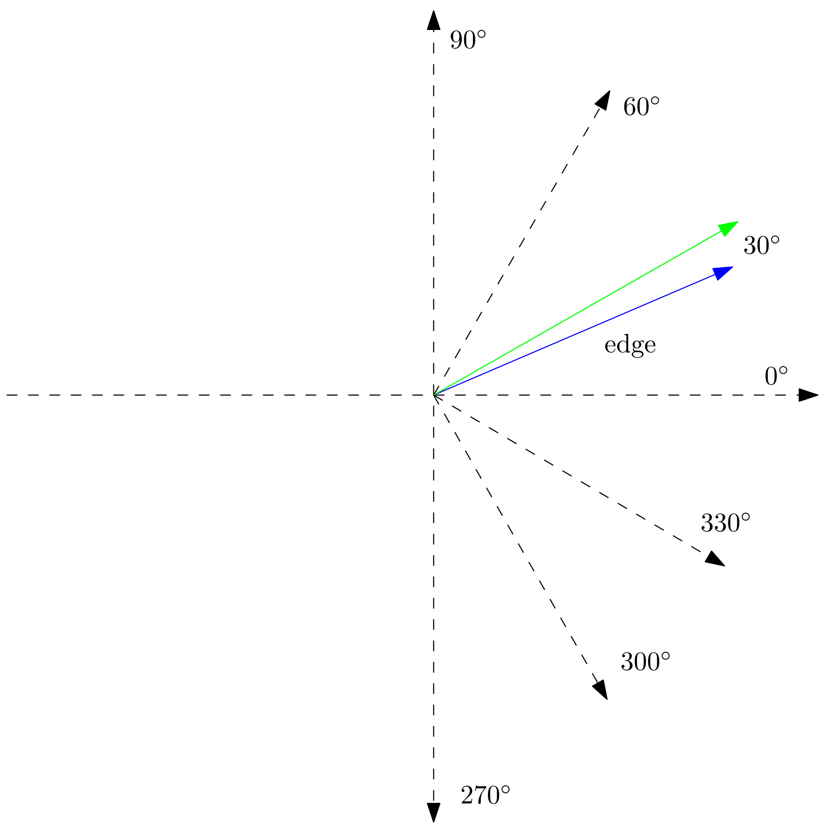


Figure 5.9: An example of restricted angles is depicted. The set of angles $\mathcal{C}_{45} = \{0°, 30°, \ldots, 360°\}$ is used.The blue edge represents a path edge. The angle depicted in green is the preferred angle of the blue edge.

5.3.2 A First Naive Approach

We show an example that illustrates why a simple greedy algorithm based on local decisions can be arbitrarily bad. Consider the example depicted in Figure 5.11(a). There are three edges of an x-monotone path.

Note that we can always connect edges to an x-monotone path, given that no two edges not share same x-coordinates. If unconnected edges that do not share (not even partially) the same x-coordinates are connected by a straight edge that connects always the two closes vertices of an edge (ties can be solved arbitrarily but no vertex can have degree 3 or more), the result is an x-monotone path. However, this technique can influence the minimum cost for the optimal solution. It is possible to connect unconnected edges in a certain way that does not influence the minimum cost for these edges. Instead of connecting the unconnected edges directly, we insert an additional vertex for two vertices we want to connect. This vertex is placed very high (i. e., above the highest horizontal strip which affects any of the unconnected edges) or very low (i. e., below the lowest horizontal strip which affects the unconnected edges). Which one of the solutions chosen is irrelevant. Then, two edges that connect each edge with the newly added vertex are inserted. Both edges are marked as v-edges. This means that we can set any horizontal strip above the ones that affect the unconnected edges to height 1. This enables us to assign the newly added edges their preferred direction.

Suppose we want to connect edge $e_i = (v_i, v_{i+1})$ and edge $e_j = (v_j, v_{j+1})$. Without loss of

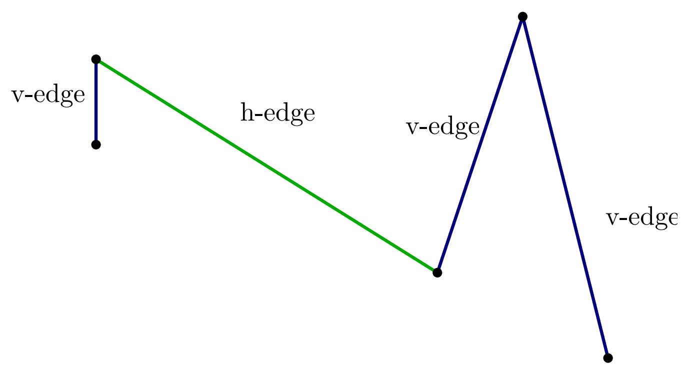
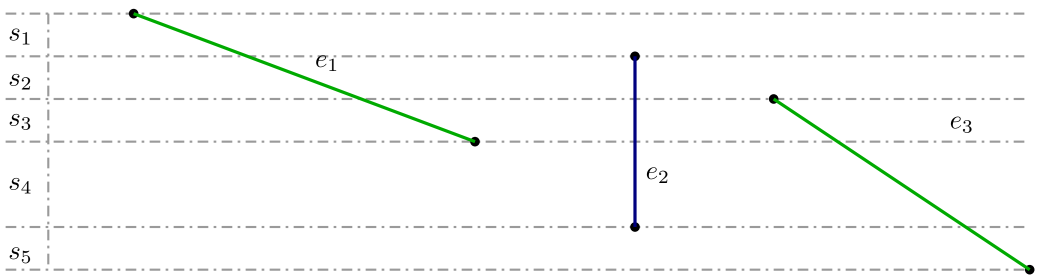


Figure 5.10: Shown are four edges. The h-edge is depicted in green and the v-edges are depicted in blue. This example cannot be solved with cost 0. Either the h-edge gets is preferred angle assigned or the leftmost v-edge. Assigning both edges their preferred angle violates the orthogonal order.

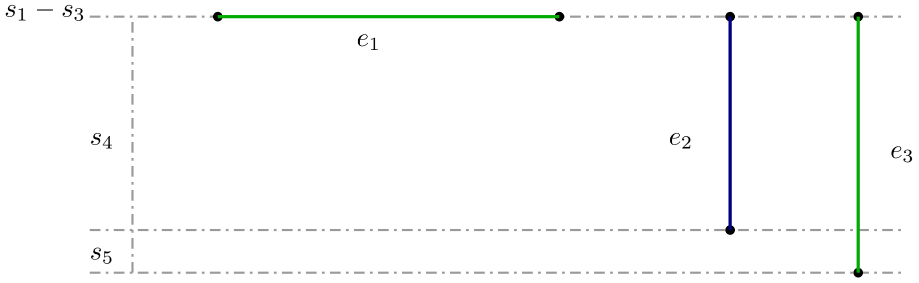
generality we assume that $x(v_i) < x(v_{i+1}) < x(v_j) < x(v_{j+1})$. If we used the first technique we would simply insert edge $e_k = (v_{i+1}, v_j)$. The second technique would need us to add vertex v_a and the two edges $e_{a1} = (v_{i+1}, v_a)$ and $e_{a2} = (v_a, v_j)$.

The edges e_1 and e_3 are h-edges and the edge e_2 is a v-edge. If we would only consider the edges e_1 and e_2 we can assign strip height values which would yield cost 0. The same holds, if we consider the edges e_2 and e_3 for any pair of edges there is a solution with cost 0. However, if we consider all three edges the situation becomes more problematic. If we want to assign e_1 its preferred (horizontal) direction, the horizontal strip heights of s_1, s_2 and s_3 must be set to 0. This "drags" the upper terminal vertex of e_2 to the same height as the edge e_1. If we want to assign e_2 its preferred direction the height of s_4 and/or s_5 have to be set to 1. This, however, pushes e_3 vertical and the result would be a cost of 1, see Figure 5.11(b). If we want both e_1 and e_3 to be assigned their preferred direction, all horizontal strip heights have to be set to 0, which again, yields cost 1 and is shown in Figure 5.11(c).

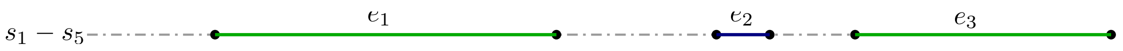
If we want to assign e_1 its preferred direction we have to either assign either e_2 or e_3 a direction which is not its preferred direction. However, this is a conflict directly caused by assigning e_1 or e_3 their preferred direction. This occurs because the edges e_2 and e_3 are entangled in this special case. One can imagine that these entanglements can become quite complex. A local decision to set a horizontal strip height to a certain value can lead to very high cost, simply because we chose to set a horizontal strip height to a certain value. Consider the example shown in Figure 5.12(a), which is very similar to that in Figure 5.11(a). The edge e_1 is a h-edge. The edges e_2 to e_4 are v-edges and the edges e_5 to e_7 are h-edges. If we set the heights of the horizontal strips which affect e_1 to 0 (see Figure 5.12(b)), we can assign e_1 its preferred direction. The strip height assignment does not directly lead to problems with the edges e_2 to e_4 or e_5 to e_7. However, as already explained in the previous example, there is a conflict. We can either assign all v-edges or all h-edges their preferred direction but not both (without violating the orthogonal order). This is done by setting the height of s_8 to 1 (to push the v-edges open) or by setting all horizontal strip heights to 0 (to set all edges horizontally). Because we decided to set the horizontal strip heights which affect e_1 to 0 we now have cost 3. The minimum cost, however, is 1, for example, by assigning $h(s_6) = 1$,



(a) Three edges of an x-monotone input path.



(b) The result of assigning $h(s_1) = h(s_2) = h(s_3) = 0$ and $h(s_4) = h(s_5) = 1$ is shown above. We cannot assign e_3 its preferred direction without violating the orthogonal order.



(c) All horizontal strip heights are set to 0.

Figure 5.11: Shown are three edges of an x-monotone path. In Subfigure (a) the original layout of the edges is shown. In Subfigures (b) and (c) different horizontal strip height assignments are shown.

which pushes the v-edges – as well as e_1 – vertical (see Figure 5.12(c)).

5.3.3 The Schematization Algorithm

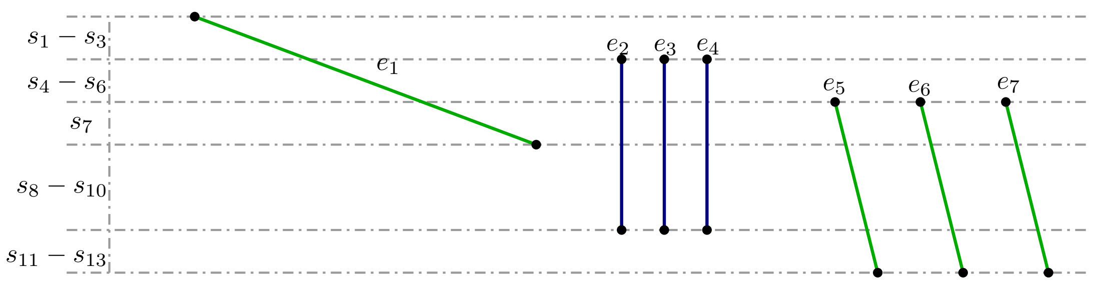
Input of the Algorithm. The input of the schematization algorithm is a polygonal path $P = (v_1, v_2, \dots, v_n)$ in the plane $\mathbb{R}^2$, where each vertex is a point $v_i = (x(v_i), y(v_i))$, $i = 1, \dots, n$ and a set $\mathcal{C}$ of allowed directions. The allowed directions are represented as angles (see Figure 4.1 as reference for the angles). We assume that $\{0, d_1, 90, d_2, 180, d_3, 270, d_4\} \subset \mathcal{C}$, where each d_i is a diagonal direction in the i-th quadrant.

Output of the Algorithm. The output of our schematization algorithm is a schematized polygonal path $Q = (q_1, q_2 \dots, q_m)$ with $m \leq n$. Every vertex $q_i \in Q$ has a corresponding vertex $v_j \in P$. The path Q maintains the orthogonal order of the input path P.

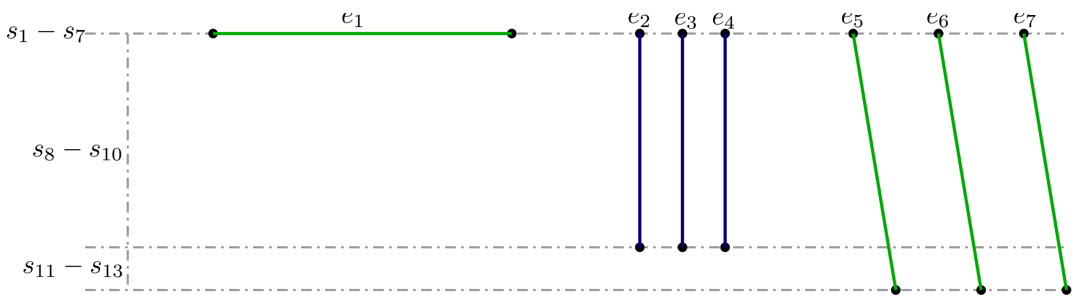
Lemma 5.3.2

By determining non-negative values for all horizontal strips s_i of an x-monotone path it is possible to assign each edge e in the schematized path P' either $\omega(e)$ or another angle contained in $\mathcal{C}$.

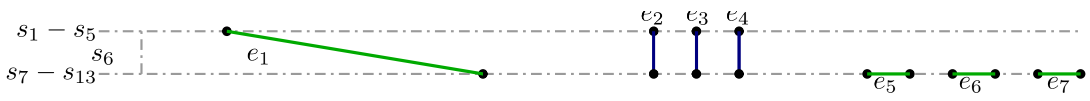
Proof: After all s_i have been assigned a value for $r(s_i)$ which is at least 0 all y-coordinates can easily be determined. We can simply traverse from the bottommost vertex to the topmost. The



(a) Edges of an x-monotone path are depicted. Note that we grouped some horizontal strips.



(b) Assigning $h(s_1)$ to $h(s_7)$ the value 0, one of $h(s_8)$ to $h(s_{10})$ and one of $h(s_{11})$ to $h(s_{13})$ the value 1 yields the results shown above. Here, e_5, e_6 and e_7 each produce cost 1. Thus, the total cost is 3.



(c) Only $h(s_6)$ gets assigned the value 1, all other horizontal strips have height 0. The total cost is 1.

Figure 5.12: In Subfigure (a) seven edges of an x-monotone path are shown. In Subfigures (a) and (a) different height assignments for the horizontal strips are shown. In all subfigures Seven are the h-eges colored in green and the v-edges are colored in blue.

lowest vertex gets assigned a fixed value for its y-coordinate (e. g.,0). Then, the next higher vertex' y-coordinate is $r(s_i) + y(v'_{i+1})$. This step is repeated until all y-coordinates have been determined.

The x-coordinates can be established by traversing from left to right. The leftmost vertex has 0 as value for its x-coordinate. For the right neighbour one of the following holds true:

- Both vertices share the same y-coordinate. An arbitrary value larger than 0 can be chosen (Recall, the angle 0° is in $\mathcal{C}$ included).
- The vertex to the right has a higher y-coordinate. A direction to the upper right is included in $\mathcal{C}$. Thus, through simple arithmetic the x-distance can be determined.
- The vertex to the right has a higher y-coordinate. A direction to the lower right is included in $\mathcal{C}$. Thus, through simple arithmetic the x-distance can be determined. ■

Explanation of the Schematization Algorithm. First, the vertices of the input path P are ordered by their y-coordinates in descending order. (The result is the sequence of vertices $U = (u_1, \dots, u_n)$.)

The core part of our algorithm is based on dynamic programming (see [Bel03, CLRS01, DL77] for an overview of the subject). See Algorithm 2 for an illustration in pseudo code.

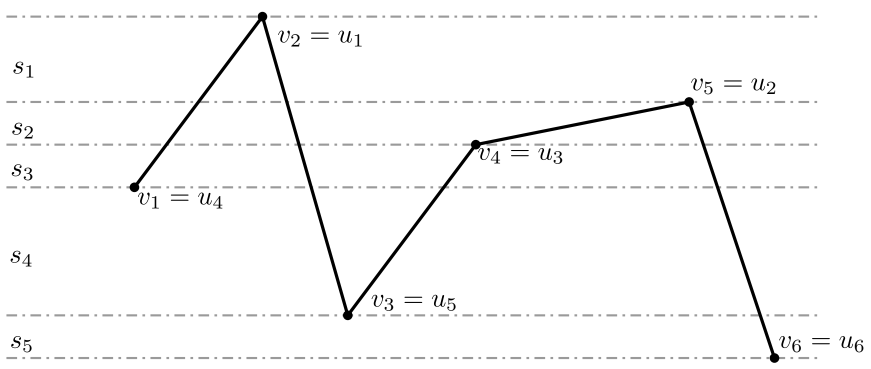


Figure 5.13: This figure shows an x-monotone path consisting of nodes v_1 to v_6. Furthermore, the horizontal strips s_1 to s_5 are depicted.

In Appendix A is a more detailed implementation of this algorithm is given. The basic idea is to recursively compare cost-minimal solutions based on combining two cost-minimal sub-solutions. Starting with the solutions for each strip s_i $(i = 1, \ldots, n-1)$ we determine and store solutions for all sets of consecutive strips $s[i,j]$, $1 \leq i < j \leq n-1$. We denote the minimum cost for the sub-instance between the vertices u_i and u_j by $OPT(i,j)$. It is important to note that while determining $OPT(i,j)$ for any i,j we only consider the edges $e = u_a, u_b$ that satisfy $i \leq a < b \leq j$ or $i \leq b < a \leq j$. In other words, edges with only one vertex between u_i and u_j are not considered.

We first determine $OPT(i,i+1)$ for $i \in \{1, \ldots, n-1\}$ and the best horizontal strip height assignment for s_i. The minimum cost is always 0. There are three cases to consider:

1. Vertices u_i and u_{i+1} are not connected by an edge. Then, we assign $h(s_i) = \ominus$ and defer the decision to a later point.

2. Vertices u_i and u_{i+1} are not connected by an h-edge. We assign $h(s_i) = 0$.

3. Vertices u_i and u_{i+1} are not connected by a v-edge. We assign $h(s_i) = 1$.

If we want to calculate $OPT(i,j)$ and $i + 1 \neq j$ there exists at least one k with $i < k < j$. We call u_k the *separating vertex*. Let $s[i,j]_0$ indicate that all horizontal strips s_l with $i \leq l \leq j$ and $h(s_l) = 0$. We define $OPT(i,j)$ to be

$$OPT(i,j) = \min_{i<k<j} \{\min\{OPT(i,k) + OPT(k,j) + c(i,k,j),\ cost(s[i,j-1]_0\}\}$$

where $c(i,k,j)$ is the minimum cost for all edges that cross u_k and that are enclosed by u_i and u_j.

After all $OPT(i,i+1)$ have been calculated, the remaining $OPT(i,j)$ have to be determined. This is done inside the two nested `for` loops (see ll. 6-7 of Algorithm 2)

Let k be any integer with $i < k < j$ while determining any $OPT(i,j)$. The algorithm constructs from two sub-solutions a new solution. This is done by the merging operation which is given in pseudo code in Algorithm 3. The algorithm first discovers the *upper limit* ul and the *lower limit* ll. The upper limit ul is the smallest horizontal strip index with $ul \in \{i, \ldots, k-1\}$ so that $\forall_{l,ul \leq l \leq k-1} h(s_l) \neq 1$. Analogously, the lower limit ll is highest horizontal strip index with $ll \in \{k, \ldots, j-1\}$so that $\forall_{l,k \leq l \leq ll} h(s_l) \neq 1$.

```
Algorithm 2: Schematization Algorithm
   Input: embedded $x$-monotone path $P = (v_1, \dots, v_n)$ with coordinates $(x(v_i), y(v_i))$
          for $i = 1, \dots, n$
   Data: table $\mathrm{OPT}(\cdot,\cdot)$, table $C(\cdot,\cdot,\cdot)$
   Output: strip height assignments $h(s_i) \in \{0, 1, \ominus\}$ for $i = 1, \dots, n-1$
1  sort nodes of $P$ in decreasing $y$-order as a sequence $U = (u_1, \dots, u_n)$
2  label each edge $e_i = (v_i, v_{i+1})$ of $P$ as either h-edge or v-edge depending on its slope
3  for $i = 1, \dots, n-1$ do
4      $\mathrm{OPT}(i, i+1) \leftarrow 0$
5      $h(s_i^{i,j}) \leftarrow \begin{cases} 0 & \text{if } (u_i, u_{i+1}) \text{ is h-edge of } P \\ 1 & \text{if } (u_i, u_{i+1}) \text{ is v-edge of } P \\ \ominus & \text{else} \end{cases}$
6  for $l = 2, \dots, n-1$ do
7      for $i = 1, \dots, n-l$ do
8          $j \leftarrow i + l$
9          $\mathrm{OPT}(i,j) \leftarrow \min_{i<k<j}\{\mathrm{OPT}(i,k) + \mathrm{OPT}(k,j) + c(i,k,j)\}$
10         update all strip heights $h(s_l^{i,j})$ for $l = i, \dots, j-1$
11 for $i = 1, \dots, n-1$ do
12     $h(s_i) \leftarrow h(s_i^{1,n}, 1, n)$
```

First, the algorithm determines the minimum cost if exactly one horizontal strip height above and exactly one horizontal strip height below the separating vertex u_k is set to 1 while all horizontal strip heights between s_p and s_q (yellow area in Figure 5.17) are set to 0. The heights of the remaining horizontal strips (green area in Figure 5.17) are not altered. Let s_p, I_q with $(ul \leq p < k \leq q \leq ll)$ be the horizontal strip heights that are set to 1. The following edges produce cost of 1(c.f. Figure 5.17).:

- All h-edges which have exactly one terminal vertex above u_p.
- All v-edges which have both terminal vertices between u_p and u_q.
- All h-edges which have on terminal vertex below u_p and one terminal vertex below u_q.

The process is repeated for every k which satisfies $i < k < j$. The minimal costs among all minimum costs is chosen as well as its corresponding horizontal strip height assignment for $OPT(i,j)$. We call the operation that determines the horizontal strip heights of the u_k crossing edges with already determined $s[i,k]$ and $s[k,j]$ the *merging* operation. If there are multiple possible horizontal strip height assignments that all produce the minimum cost any of them can be chosen arbitrarily.

Then, the algorithm determines the minimum costs if all horizontal strips heights between s_k and s_{ll} are set to zero and only one horizontal strip height above u_k (i. e., one of the horizontal strip s_{ul} to s_{k-1}) is set to 1. All s_v with $p < v < k$ are set to zero. The heights of all horizontal strips above s_p remain the same. Let s_p with $(ul \leq p < k)$ be the horizontal strip height that is set to one. The following edges each produce cost 1 (see Figure 5.15). Note that every edge has to be a u_k crossing edge. Otherwise the edge would have already been considered either by $OPT(i,k)$ or $OPT(k,j)$:

- All h-edges with exactly one terminal vertex above u_l and one terminal vertex below u_p (i. e., below u_k, because the edge has to be u_k crossing).
- All v-edges with both terminal vertices below u_p and above u_{ll}.

Then, the algorithm calculates the minimum cost if all horizontal strip heights above the separating vertex u_k are set to zero only one horizontal strip below u_k is set to one. All s_v with $k < v < q$ are set to zero. The heights of all horizontal strips below s_p remain the same. Let s_q with $(k \leq q \leq ll)$ be the horizontal strip height that is set to one. The following edges produce cost of 1 (c.f. Figure 5.16).:

- All h-edges with exactly one terminal vertex below u_q and one terminal vertex below u_q.
- All v-edges with both terminal vertices above u_q (one of them needs to be above u_k and one below u_k).

The algorithm then calculates the cost if all horizontal strips between the upper and lower limit have height 0.

Finally, the algorithm calculates the cost if all horizontal strips are which are between u_i and u_j are set to 0. The cost equals the number of v-edges which lie completely between u_i and u_j. If all horizontal strips are set to 0 the edge must be drawn horizontal and this produces cost of 1 for each v-edge. The h-edges can be drawn horizontal without producing any cost. Note that this can only happen if every horizontal strip s_l, $i \leq l < j$ affects at least one h-edge.

If there are multiple optimal solutions which produce each the same cost, then the solution which sets to horizontal strip heights to one is preferred other a solution where only one horizontal strip height is set to one. The least preferred solution is the solution where all horizontal strip heights are set to 0. If there are multiple solutions of the same type the solution is preferred which places the horizontal strips with height 1 closer to the separating vertex is preferred. This distinction ensures that if there is a horizontal strip s_l with $h(s_l) = 0$ there is a h-edge which is affected by s_l an can be drawn horizontally.

This technique is repeated until $OPT(1, n)$ has been determined. The algorithm stops and the optimal horizontal strip heights are returned.

Correctness. After describing the algorithm in detail we prove its correctness. In particular the orthogonal order is not violated and the cost of the solution is minimum over all possible horizontal strip height assignments. It is easy to see that by assigning a non-negative height to each strip the y-order of the nodes of P does not change. Since we draw P in an x-monotone manner this implies that the orthogonal order of all nodes is preserved. The proof that the algorithm produces a horizontal strip height assignment of minimal cost is more difficult. We have to prove that the merging operation computes a minimum-cost solution. does indeed choose the minimum cost. For that, we have to prove that there is no possibility that the cost of $OPT(1, n)$ is not optimal, because we have set a horizontal strip height during the computation of $OPT(i, j)$ to a value which turns later out to be the wrong choice for $OPT(1, n)$. An example is given in Figure 5.18.

Algorithm 3: Merge Operation

```
    Input: Horizontal strip heights $s[i, j]$
    Output: Altered horizontal strip heights $s'[i, j]$, minimum cost $min$
 1  for $k \leftarrow i + 1, \ldots, j - 1$ do
 2      Determine upper and lower limit $ul$ and $ll$
 3      $min = \infty$
 4      for $u \leftarrow k - 1, \ldots, ul$ do
 5          for $l \leftarrow k, \ldots, ll$ do
 6              $s'[i, j] = s[i, j]$
 7              $h(s'_u) = 1$
 8              $h(s'_l) = 1$
 9              for $z \leftarrow u + 1, \ldots, l - 1$ do
10                  $h(s'_z) = 0$
11              $\mathcal{C}$ = Calculate $cost(s'[i, j])$
12              if $c > min$ then
13                  $min = c$
14  $c$ = Calculate $cost(s'[i, j]_0)$
15  if $c > min$ then
16      $min = c$
```

Lemma 5.3.3

For any given x-monotone path P assigning each $h(s_i)$ a non negative value does not violate the orthogonal order.

Proof: Let P' the path which results if horizontal strips are all equal to or larger than zero (i. e.,we change the y-coordinates of the vertices in P and the result is P'). Assume that there are two vertices $v_1, v_2 \in P$ and their corresponding vertices v'_1, v'_2 in P'. Further, these two vertices violate the orthogonal order in P'. Because we did not change the x-coordinates this means that w.l.o.g. $y(v1) \leq y(v2)$ and $y(v'_1) > y(v'_2)$ hold true. But if v_1 is above v_2 then there has to be at least one horizontal strip between v_1 and v_2. Even if all of them are set to zero the orthogonal order would not be violated. If only one horizontal strip has a height larger than 0, $y(v1') < y(v'_2)$ would hold true. Obviously, only horizontal strips with a negative height can result in a violation of the orthogonal order. ■

We show that the merging operation determines for any $OPT(i, j)$ and any k, $i < k < j$ the best horizontal strip heights (i. e.,yielding minimal costs) for all u_k crossing edges.

Lemma 5.3.4

The operation for computing $c(i, k, j)$ for all $i < k < j$ calculates a height assignment for all horizontal strips between s_{ul} and s_{ll} with minimum cost.

Proof: Let $i < k < j$ be the indices of three nodes of P and suppose we want to compute $c(i, k, j)$ in order to determine $OPT(i, j)$. We need to show that $c(i, k, j)$ is indeed the minimum cost of all edges that cross u_k. For sake of contradiction suppose there is a height assignment with cost $c_{min} < c(i, k, j)$.

Consider what the operation does to determine the minimum cost. First, it tests what the cost is if all horizontal strip heights between s_{ul} and s_{ll} are set to 0. Then, it determines the

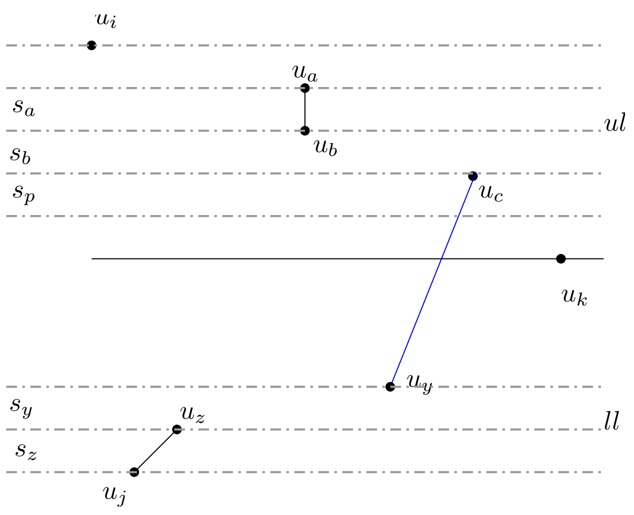


Figure 5.14: This figure shows an example of nodes of the path P (we use the alternative notation of the nodes from the ordered sequence U.) Here, $h(s_a) = h(s_z) = 1$. The upper limit is b and the lower limit is s_y. The blue edge is a u_k crossing edge.

minimum cost if only one horizontal strip height above u_k from s_{k-1} to s_{ul} is set to 1 and all below u_k are set to 0. The operation then, calculates the minimum cost if one horizontal strip height of the horizontal strips between s_k and s_{ll} is set to 1 and all other horizontal strips are set to 0. Finally, it calculates the minimum costs if one horizontal strip height between s_{ul} and s_{k-1} and one between s_k and s_{ll} is set to 1 and all others are set to 0. This means that any horizontal strip height assignment that is not considered by the merging operation has to set at least two horizontal strips heights between s_{ul} and s_{k-1} or s_k and s_{ll} to one. We now prove that this cannot yield lower cost. Thus, the assumption is wrong and the merging operation does indeed determine the minimum cost.

There is a horizontal strip height assignment that sets at least two horizontal strip heights between s_{ul} and s_{k-1} to one. Further, let there be zero or more horizontal strip heights set to one between s_k and s_{ll}. Let s_d be the first horizontal strip above u_k with a height that is set to one. Let s_e be any other horizontal strip above s_d which height has been set to one (see Figure 5.19 for an illustration).

- First, let's assume that there is no horizontal strip between s_k and s_{ll} set to one. Recall how the cost is determined (see Section 5.3.3). Note that the cost consists of the number of h-edges with one terminal vertex above s_d and one below u_k. This means that the value of s_e has no influence on the costs produced by h-edges, because the horizontal strip s_d already pushes the h-edges vertical. Further, costs are produced by v-edges that have one terminal point below s_d and one below u_k. These are the same cost as described in (see Section 5.3.3). This cost is not influence by horizontal strips above s_d.
- Second, lets now assume that there is at least one horizontal strip between s_k and s_{ll} with a height set to 1. Let this horizontal strip be called s_f. The costs are caused by all h-edges which have either one terminal vertex above s_d and one below u_k or one terminal vertex above u_k and one below s_f. Additionally, every v-edge that has both terminal vertices below u_d and above u_{f+1} produces cost of 1. As we see again, the horizontal strip s_e is completely irrelevant. If any h-edge is pushed open by s_e it is already pushed open by s_d (recall, all edges the merging operation considers have to be u_k crossing). Further, the horizontal strip height of s_e has no influence on the cost of the v-edges which are set to the horizontal direction.

The proof for the case that there are at least two horizontal strips between s_k and s_{ll} with

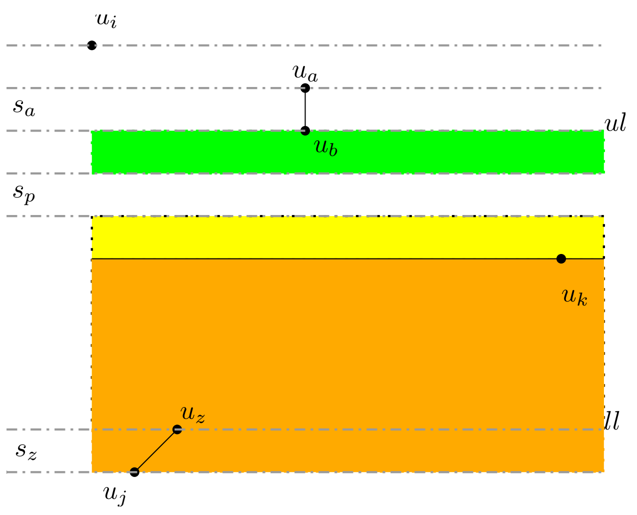


Figure 5.15: This figure illustrates how the cost is determined for $OPT(i, j)$ if exactly one horizontal strip above the separating vertex u_k is set to height 1. All horizontal strips in the yellow area have height 0, all horizontal strips in the green area remain untouched. Further, all horizontal strips between u_k and u_z are set to 0.

heights set to 1 and zero or more horizontal strip heights between s_{k-1} and s_{ul} that are set to 1 follows from a symmetric argument. This means that $c(i, k, j)$ indeed equals the minimum cost attainable by setting the horizontal strip heights between s_{ul} and s_{ll}. ■

The following lemma helps us prove the correctness of the algorithm.

Lemma 5.3.5

Assigning the value $h(s_i) = 1$ to any s_i, $1 < i < n$ determines for all edges that cross s_i if they produce cost or not.

Proof: Each s_i crossing h-edge is pushed vertical, and hence produces cost of 1. All crossing v-edges can be assigned their preferred direction. Thus, no v-edge that crosses s_i produces cost. ■

Now that we have proven that the $c(i, k, j)$ operation calculates the minimum cost for any $OPT(i, j)$ and a given k, $i < k < j$ we have to prove that the algorithm correctly calculates $OPT(1, n)$. Recall that the merging operation only computes $c(i, k, j)$, the minimum cost for the u_k crossing edges. It does not compute the minimum cost for the remaining edges.

Theorem 5.3.1

The presented schematization algorithm (Algorithm 2) determines a horizontal strip height assignment that produces minimal cost among all possible horizontal strip height assignments.

Proof: We prove the theorem by induction.

Induction hypothesis: We maintain the invariant that if for any s_i the value $h(s_i) = 0$ there has to be at least one h-edge e that is affected by s_i and is drawn horizontally, i. e.,with cost 0.

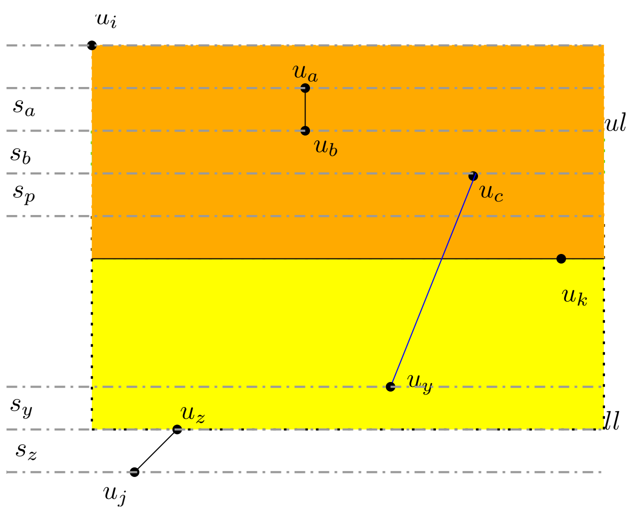


Figure 5.16: This figure illustrates how the cost is determined for $OPT(i,j)$ if exactly one horizontal strip below the separating vertex u_k is set to height 1. All horizontal strips in the yellow area have height 0, all horizontal strips between u_b and u_k have height 0, too.

Begin of induction: $OPT(i,j)$ is the minimum cost solution for $j = i+1$ and $1 \leq i < n$.

There are only three cases which have to be evaluated (see ll. 3-5 in Algorithm 2):

1. The vertices u_i and u_{i+1} are vertices of the same h-edge. The horizontal strip height is set to 0 and the minimum cost is 0.
2. The vertices u_i and u_{i+1} are vertices of the same v-edge. The horizontal strip height is set to 1 and the minimum cost is 0.
3. The vertices u_i and u_{i+1} are not part of the same edge. The horizontal strip height is set to $\ominus$ and the minimum cost is 0.

Obviously, the minimum cost of any $OPT(i, i+1)$ is 0. The algorithm does calculate the minimum cost of $OPT(i, i+1)$ correctly. Only in one case $h(s_i)$ is set to 0. This happens only if u_i and u_{i+1} are part of the same h-edge and thus the invariant holds.

Induction step: Let $i < j$ and $j - i = k$. We may assume by the induction hypothesis that $OPT(a,b)$ for any $a < b, b-a < k$ has been computed correctly and that for all s_i with $h(s_i) = 0$ the invariant holds.

Let $\mathcal{L} = \{L_1, \ldots, L_l\}$ be the set of optimal solutions for $OPT(i,j)$. Let $s^g[i,j]$ be the horizontal strip height assignment of L_g. Let $MaxOne(L_g) = \max\{f \mid h(s^g_f[i,j]) = 1, i \leq f \leq j\}$. Further, let $cost(s^g[i,j])$ be the cost produced by the horizontal strip height assignment. Note that for any $k, i < k < j$ and for any $L_g \in \mathcal{L}$ the following holds by the induction hypothesis: $OPT(i,k) \leq cost(s^g[i,k])$ and $OPT(k,j) \leq cost(s^g[k,j])$. Note that during the proof we make no distinction between the directed edge (u_i, u_j) and (u_j, u_i).

Case 1: The set $\mathcal{L}$ contains a single solution that is setting the heights of all horizontal strips to 0. Our algorithm checks the cost if all horizontal strips are set to 0. This cost is considered and for a solution, where that is the optimal solution the algorithm will find it.

In the following, let $L_z = \arg\max\{MaxOne(L_g) \mid L_g \in \mathcal{L}\}$.

Case 2: There is at least one solution in L that contains a horizontal strip set to height 1. In this case we choose the solution $L_z \in \mathcal{L}$ that has among all horizontal strip height assignments the one with the largest index k for which $h(s_k) = 1$.

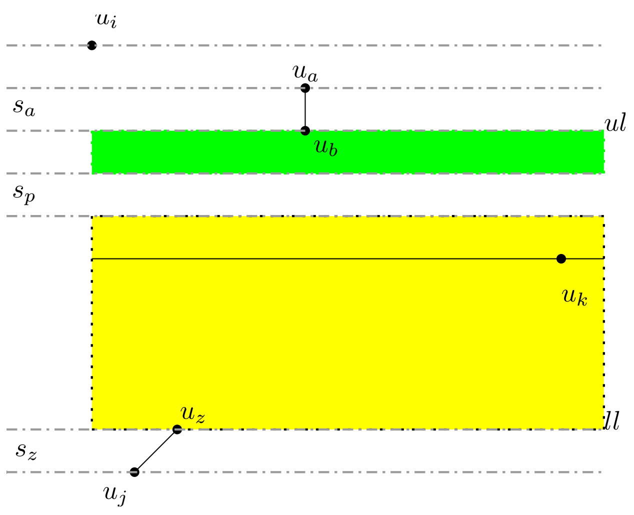


Figure 5.17: This figure illustrates how the cost is determined for $OPT(i,j)$ if exactly one horizontal strip above the separating vertex u_k and exactly one below u_k is set to height 1. All horizontal strips in the yellow area have height 0, all horizontal strips in the green area remain untouched.

Case 2a: strip s_k is the bottommost horizontal strip s_{j-1} in $OPT(i,j)$. We consider the case that u_{j-1} is the separating vertex. Due to our induction hypothesis $OPT(i,k)$ and $OPT(k,j)$ have already been determined and produce minimum cost.

We distinguish three cases. If in $OPT(j-1,j)$ the algorithm assigned $h(s_{j-1}) = 0$, this implies that there is a h-edge $e = (u_j, u_{j-1})$. The solution L_z pushes this edge vertical and hence produces cost of 1. However, L_z might push this edge vertical in order to push another v-edge (u_l, u_j) vertical, too. This produces at least the same cost as our solution (Recall $OPT(i, j-1)$ has already been determined to be optimal).

If our algorithm assigns $h(s_{j-1})$ the value 1 there is a v-edge $e = (u_j, u_{j-1})$. Because $OPT(i, j-1)$ is an optimal solution and one optimal solution L_z has $h(s_{j-1}) = 1$, our solution is optimal, too.

If our algorithm assigns $h(s_{j-1})$ the value $\ominus$ there is no edge (u_j, u_{j-1}). Our algorithm will consider the cost produced by setting $h(s_{j-1}) = 1$. Because $OPT(i, j-1)$ is optimal and one optimal solution has set $h(s_{j-1}) = 1$, an optimal solution is determined by our algorithm.

Case 2b: Strip s_k is not the bottommost horizontal strip in $OPT(i,j)$. We consider the case that u_{k+1} is the separating vertex. Due to our induction hypothesis $OPT(i, k+1)$ and $OPT(k+1, j)$ have already been determined and produce minimum cost.

One optimal solution has determined $h(s_k)$ to be 1. Our algorithm assigns $h(s_k)$ the value 1 or $\ominus$. We proved in Lemma 5.3.5 that for all s_k crossing edges the cost are fixed if $h(s_k) = 1$. Note that the cost of the optimal solution L_z can be decomposed into the cost for all edges enclosed by u_i and u_k, the cost for all edges enclosed by u_{k+1} and u_j, and the cost of all edges crossing s_k. Hence any solution that is optimal for the edges enclosed by u_i and u_k, as well as, for those enclosed by u_{k+1} and u_j is optimal for the whole instance if we set $h(s_k) = 1$. We show that our algorithm considers this case. This is the case if it has already assigned $h(s_k) = 1$ in $OPT(i, k+1)$ or it has assigned $h(s_k) = \ominus$ and sets $h(s_k) = 1$ during the merging step. Thus, we now only consider cases where our algorithm has determined that $h(s_k) = 0$. This occurs only if there is an h-edge incident to u_{k+1} which is affected by s_k and can be drawn horizontally if $h(s_k) = 0$ (see invariant).

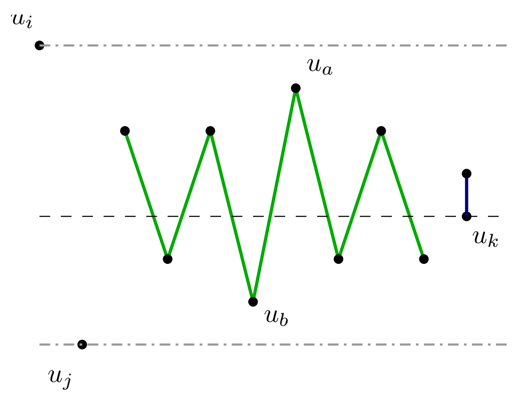


Figure 5.18: The green edges are h-edges and the blue edge is a v-edge. The vertex u_k is the separating vertex. If the v-edge is assigned its preferred direction the cost is very high because the h-edges are pushed open.

Vertex u_{k+1} is incident to a single edge e. This edge has to be the aforementioned h-edge. In the optimal solution L_z the value of $h(s_k)$ is 1. However, if we would change in L_z the value of $h(s_{k+1})$ from 0 to 1 the same cost would be produced. Every h-edge that is pushed vertical by $h(s_{k+1})$ is already pushed vertical by $h(s_k)$. Every v-edge pushed vertical by $h(s_k)$ is also pushed vertical by $h(s_{k+1})$. Thus, there exists a solution L_z^* with $MaxOne(L_z^*) > MaxOne(L_z)$. This is a contradiction to our initial assumption.

Vertex u_{k+1} is incident to two edges e_1 and e_2 that both have u_{k+1} as their lower endpoint. All edges that are pushed vertical by $h(s_k)$ are also pushed vertical if $h(s_{k+1}) = 1$ is chosen and there is no edge (u_c, u_{k+1}) with $y(u_c) < y(u_{k+1})$ since the degree of each vertex is at most two. Hence, this does not increase the cost of the solution. Thus, there exists a solution L_z^* with $MaxOne(L_z^*) > MaxOne(L_z)$. This is a contradiction to our initial assumption.

Vertex u_{k+1} is connected by two edges e_1 and e_2, one of which has u_{k+1} as their lower and the other as upper endpoint. Let $e_1 = (u_a, u_{k+1})$ and $e_2 = (u_b, u_{k+1})$, where $i \leq a < k+1 < b \leq j$. If e_2 is a v-edge setting $h(s_{k+1}) = 1$ pushes all those edges vertical that cross u_k. These edges are already pushed vertical due to $h(s_k) = 1$. Furthermore, this solution reduces the cost by one which contradicts the optimality of L_z.

If e_2 is an h-edge we know that it produces no cost in solution L_z. In our solution the h-edge e_1 produces no cost (see invariant). Otherwise $h(s_k)$ would not have been set to 0 by our algorithm. Consider the height assignment in L_z. If we change $h(s_k)$ to 0 and $h(s_{k+1})$ to 1 the same u_{k+1} crossing edges are pushed vertical. Further, e_2 is pushed vertical, producing cost of 1. However, now we could produce a solution L_z^* that consists of the solution produced by $OPT(i, k+1)$, combined with $h(s_{k+1}) = 1$ and $s^z[k+2, j]$. The solution produces cost 1 for edge e_2 but edge e_1 is drawn horizontally, thus saving cost 1. Hence, this solution produces the same cost as L_z but with $MaxOne(L_z^*) > MaxOne(L_z)$. This is a contradiction to our initial assumption. We were able to show that in any case our algorithm either is able to determine an optimal solution for any $OPT(i, j)$. ■

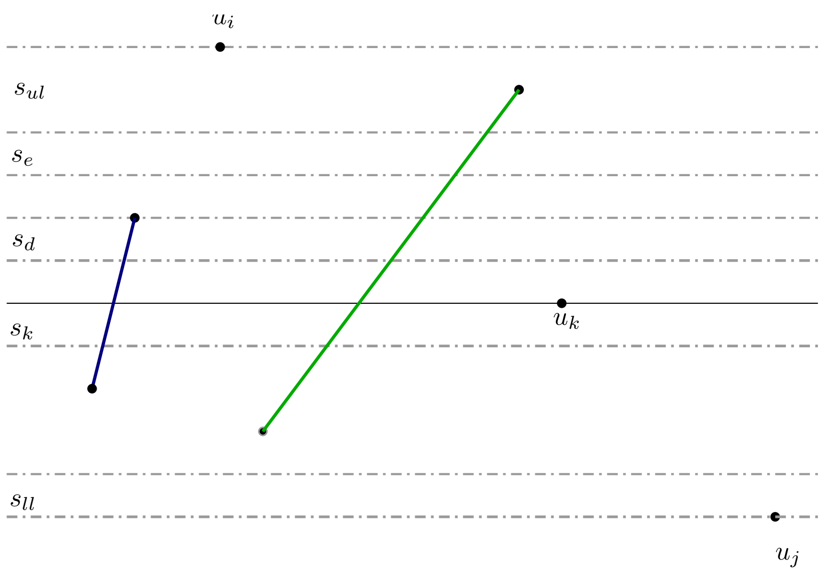


Figure 5.19: This figure illustrates that setting $h(s_e)$ and $h(s_d)$ both to 1 cannot decrease the cost. Neither for the (green) h-edge nor for the blue $v - edge$.

Time complexity. In the following we discuss the running time of algorithm 2.

Lemma 5.3.6

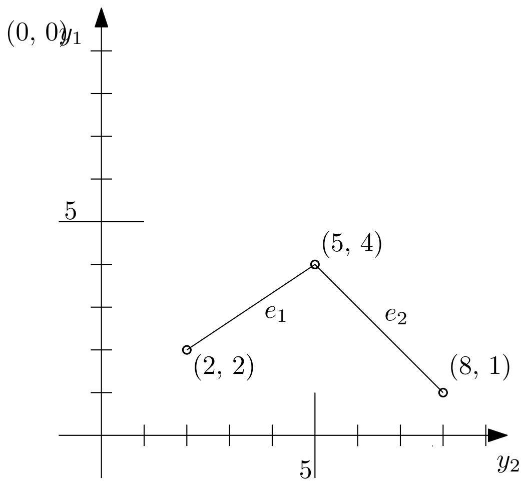
Let n be the number of vertices of the x-monotone input path P. The asymptotic time complexity of Algorithm 2 is in $O(n^6)$ and requires $O(n^3)$ space.

Proof: In the beginning the algorithm sorts the vertices of path P in ascending order. This can be done in $O(n \log n)$. It calculates the values for every $OPT(i, j)$. These are $O(n^2)$ values. For each we consider $O(n)$ possible values k to determine the splitting vertex v_k. During each merging step it calculates

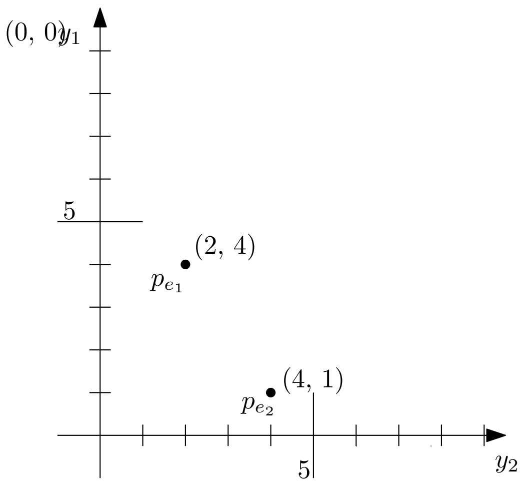
1. The cost if all horizontal strip heights are set to 0 ($O(n)$).
2. The cost if one horizontal strip height above the separating vertex u_k is set to one 1 and all other are set to 0 ($O(n)$).
3. The cost if one horizontal strip height below the separating vertex u_k is set to one 1 and all other are set to 0 ($O(n)$).
4. The cost if exactly one horizontal strip height above and one below is set to 1 and all horizontal strip heights between them are set to 0. This alone are $O(n^2)$ iterations. However, to calculate the cost we have to determine which edges are u_k crossing and which one of those produce costs. This takes another $O(n)$ for each iteration. Yielding a running time of this part of the algorithm of $O(n^3)$.

Unfortunately, we have to repeat the steps 2-4 for every $OPT(i, j)$ yielding $O(n^2)$ running time. We have to calculate for every k with $i < k < j$ the costs yielding another $O(n)$ iterations. Determining the cost is dominated by step 4 which lies on $O(n^3)$. Thus, we have a total running time of $O(n^6)$.

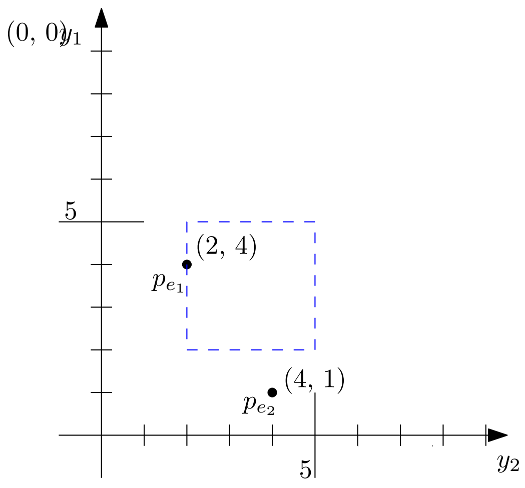
Because we store for each $OPT(i, j)$ the optimal horizontal strip height assignment we need storage of $O(n^3)$. ■



(a) This figure shows three vertices and the two edges e_1 and e_2 that connect them.



(b) The edges from Subfigure (a) are transformed into points.



(c) A range query for all edges between y-coordinates 2 and 5 is carried out. All points on inside and on the rectangle with corner nodes $\{(2,2),(2,5),(5,5),(5,2)\}$ are returned.

Figure 5.20: This figures show how to use a range tree to determine how many edges are between to given y-coordinates.

Running time Improvement. Unfortunately, the running time of the Algorithm 2 is quite high. But optimizations are somewhat limited if we do not change the algorithm drastically. We need at least $O(n^2)$ for every $OPT(i, j)$ iteration and additionally $O(n)$ for every k with $i < k < j$. Thus, the only way to significantly reduce the algorithms running time is by reducing the time needed for calculating the minimum cost in step 4. If we look at the algorithm 2 we see that it calls Algorithm 11 $O(n^2)$ times. The main problem is to determine which edges to consider. For that we can use range trees (see [GO97]). The traditional use of range trees is to extract a set of vertices in a plane that all lie in the same rectangle. The construction of a range tree can be done in $O(n \log n)$ time and for a given rectangle region we can extract all vertices in this rectangle in $O(\log n)$. We need $O(n \log n)$ storage for a range tree that stores n vertices.

For our purpose we build two different range trees. One of them stores the h-edges and one stores the v-edges. An edge is inserted as point with the x-coordinate set to the value of one terminal vertices y-coordinate. And the y-coordinate is set to the value of the other terminal vertices y coordinate. Thus, we simply identify an edge by its two y-coordinates. In Figure 5.3.3 an example is shown which illustrates how range trees can be used for storing the edges. During the execution of the Algorithm 3 it needs to know how many v-edges are not pushed vertical and how many h-edges are pushed vertical to determine the cost. To do that we build the two range trees as described during the merging operation for each value of k. However, we only insert u_k crossing edges. The construction cost is $O(n \cdot (n \log n))$, because we have to look at at most n vertices to determine the h- and v-edges. During each iteration of algorithm 11 we can now very easily determine the costs. We simply have to calculate how many h-edges are between i and j and how many h-edges are between the upper horizontal strip with height set to 1 and the lower horizontal strip with height set to 1 for an illustration). Both operations are possible in $O(\log n)$. We have to do the same thing for the v-edges. But we only need to identify the number of v-edges which lie between the upper horizontal strip with the height 1 and the lower horizontal strip with height 1. This operation is in $O(\log n)$ too. With this operation we are able to reduce the asymptotic running time to $O(n^5 \log n)$. A more detailed analysis may yield a lower asymptotic running time.

There is another idea to improve the running time of the algorithm which unfortunately does not improve the asymptotic running time without. Under certain conditions the complicated schematization Algorithm 2 need not be applied to all nodes. An algorithm which uses the schematization algorithm does not necessarily need to compute $OPT(1, n)$ but only parts. The general idea is that if there are consecutive horizontal strips which can only affect v-edges, the horizontal strip heights of those can be set to 1. If there is a horizontal strip which affects at least one h-edge the schematization Algorithm 2 has to be called. This idea is shown in Algorithm 4. It is obvious that the asymptotic running time cannot be improved because we can not guarantee that the algorithm ever can use this idea. However, there might be cases where this can yield a decrease in necessary computing time.

Calculating the true horizontal strip height values. After Algorithm 2 has been completed we have a set of intervals with a height of either 0 or 1. Recall that the height of 1 only symbolizes that the height is larger than 0. Every horizontal strip height with height 1 can be set to any positive value and every horizontal strip with height 0 remains at height 0. We only have to determine the height of those horizontal strips which are set to 1. As stated above in Section 5.3.1 we not only want a minimum number of edges which cannot be assigned

Algorithm 4: `SchemSmallImprovment`

```
Input: embedded $x$-monotone path $P = (v_1, \dots, v_n)$ with coordinates $(x(v_i), y(v_i))$
       for $i = 1, \dots, n$
Data: table OPT$(\cdot,\cdot)$, table $C(\cdot,\cdot,\cdot)$
Output: strip height assignments $h(s_i) \in \{0, 1, \ominus\}$ for $i = 1, \dots, n-1$
1  $U \leftarrow$ sort$(P)$
2  $n \leftarrow |P|$
3  start $\leftarrow$ 0
4  end $\leftarrow$ -1
5  for $i \leftarrow 1$ to $n$ do
6      if $u_i$ and its left or right neighbour form a h-edge then
7          end $\leftarrow$ i
8          set all horizontal strip heights between $s_{start}$ and $s_{end}$ to 1
9          hcount $\leftarrow$ 1
10         while $hcount > 0$ do
11             i++
12             if $u_i$ and its left neighbor $u_l$ form a h-edge then
13                 if $u_l$ is above $u_i$ then
14                     hcount$--$
15                 else
16                     hcount++
17             if $u_i$ and its right neighbor $u_r$ form a h-edge then
18                 if $u_r$ is above $u_i$ then
19                     hcount$--$
20                 else
21                     hcount++
22         Schematize vertices between $u_{end}$ and $u_i$ with Algorithm 2
23         start $\leftarrow$ end
24 if $start \neq i$ then
25     set all horizontal strip heights between $s_{start}$ and $s_i$ to 1
```

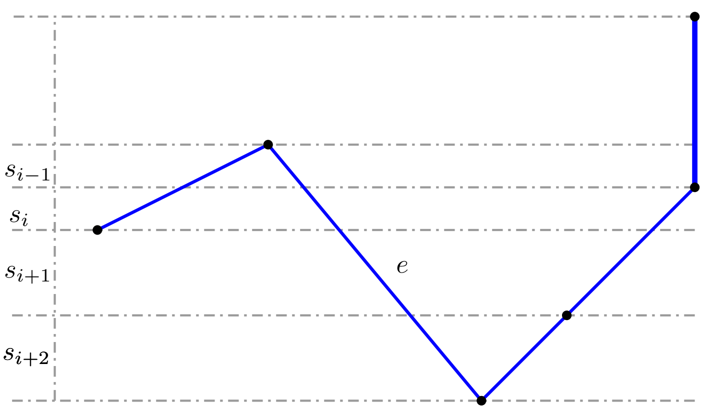


Figure 5.21: If we want to minimize the total path length (the complete path is highlighted in blue) we have to consider the horizontal strip heights differently. Consider the horizontal strip depicted in red. Its horizontal strip height influences three different horizontal strip heights.

their preferred direction but the resulting path length should be minimal. We formulate this problem as a linear programming (LP) problem. The theory behind linear programming has been widely studied and analysed. For an introduction see [CLRS01]. For more details and examples see [Sch98, AF98] and [dBvKOS00].

First, we illustrate how to formulate an LP with the goal that every edge in the schematized path has at least the length of its corresponding edge in the input path. At the same time the overall path length is to be minimized. Later we show how to change one part of the LP[1] to introduce a minimum path length.

There may be one or several horizontal strips between the two vertices of any edge.

The path length that is to be minimized is calculated by calculating the total sum for all edges of the true horizontal strip heights that affect the edge. In the first formulation we determine the original edge length and determine the height all horizontal strips have to have so that the edge in the schematized path has at least this length.

In the following let $e' = (v'_i, v'_{i+1})$ be an edge in a schematized path. Let $e = (v_i, v_{i+1})$ be the corresponding edge in the original path. Further, let $\alpha_{e'}$ be the preferred angle assigned to e'. The following notations are important:

$$\Delta_e = \sqrt{(y(v_i) - y(v_{i+1}))^2 + (x(v_i) - x(v_{i+1}))^2} = \|\overline{v_i v_{i+1}}\|$$

$$\Delta_{x'_e} = \cos \alpha_{e'} \cdot \Delta_e$$

$$\Delta_{y'_e} = \sin \alpha_{e'} \cdot \Delta_e$$

We denote the true (i. e., non-symbolic) height of a horizontal strip s_i by $r(s_i)$.

The aim of the LP is to

$$\text{minimize} \sum_{e'=(v'_i,v'_{i+1})\in P'} \sum_{s_j \text{ affects } e'} r(s_j)$$

[1]To solve out LP we used the open source library *lpsolve* (see http://sourceforge.net/projects/lpsolve/).

subject to

$$\sum_{s_j \text{ affects } e', h(s_j) \neq 0} r(s_j) \geq \Delta_{y_{e'}} \, \forall_{e'=(v'_i, v'_{i+1}) \in P'}$$

We cannot simply subject the constraints to minimize the sum of all interval heights. This does not equal the path length. Consider the example depicted in Figure 5.21. Only minimizing the total height of all strips does not yield a minimum path length. In the example mentioned, the horizontal strip s_i affects multiple edges. Thus, we have to minimize the horizontal strip heights according to the edges it affects. We have to "travel" along the edges and add up the length of each edge. The length of each edge is influenced by the height of the horizontal strips which affect the edge and by its preferred direction. If we travel from the leftmost to the rightmost vertex we can determine the length of any edge by calculating the $\sin(\alpha)$, where α is the angle of the direction of the edge. The length of the hypotenuse is the the sum of the horizontal strip heights divided by $\sin(\alpha)$.

If we want to introduce a minimum edge length min_length the Δ_e changes to

$$\Delta_e = \min\{\sqrt{(y(v_i) - y(v_{i+1}))^2 + (x(v_i) - x(v_{i+1}))^2}, \text{min\_length}\}$$

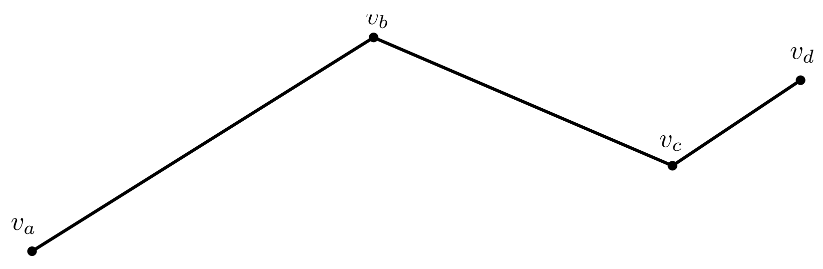
The remaining formulation of the LP stays the same. With this little addition we ensured that every segment of the path has at least a certain length. Because in the final output we scale the complete path so that it fits unto one DIN A4 sheet of paper other edges are possibly shortened (e. g.,segments of highways are prime candidates).

Finding the actual coordinates. After the true heights of each horizontal strip has been determined we need to join the vertices to a path. The calculation of the horizontal strip heights has been done either by Algorithm 2 for the horizontal strip heights which are set to 0 or by solving the previously explained LP. Joining the vertex to a path is fairly easy as all the relative y-coordinates of all vertices are already fixed by the height of all horizontal strips. Recall, we deal only with x-monotone paths. This means that we can set the x-coordinates very easily. The joining process is illustrated in Algorithm 5.

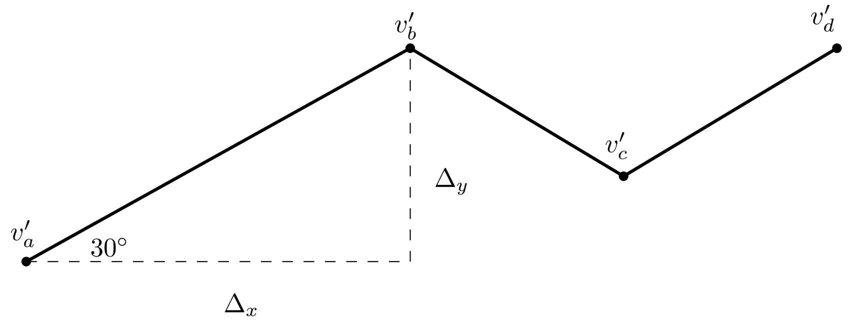
The basic idea is to first sort the vertices of the path P again in descending order by their y-coordinates. Then, for every vertex the y-coordinate is determined in $O(n)$ by traversing from bottom to top. The smallest y-coordinate starts at some fixed value. Finally, the x-coordinates are determined by visiting each vertex from left to right. An example how to determine the x-coordinate is shown in Figures 5.22(b).

The running time of the algorithm is dominated by sorting the vertices by their y-coordinate in descending order. Determining the x- and y-coordinates each is possible in $O(n)$ time. Thus, the total running time is in $O(n \log n)$.

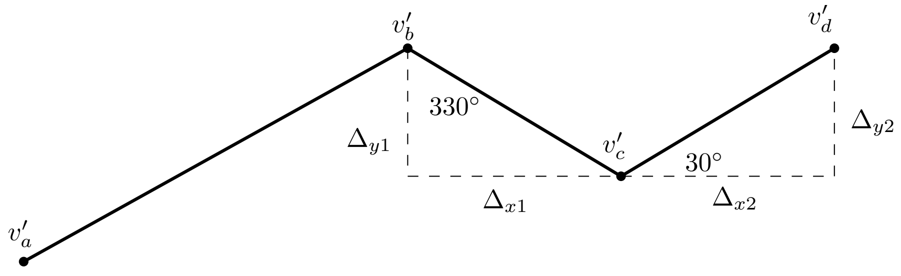
Slight Modification. We presented an algorithm that is able to set the horizontal strip heights to an appropriate value that does not violate the orthogonal order. Further, we illustrated how the absolute x- and y-coordinates can be determined. However, the algorithm in its current form can lead to results which are undesirable. An example is shown in Figure 5.23. Such a result can happen if only few horizontal strip heights are set to one by the main Algorithm 2. In the introduction we established the rule that any horizontal strip height set to 1 implies that the horizontal strip height has to be larger than 0. We can now relax this rule. We now say that any horizontal strip height which is set to 0 by Algorithm 2 stays at height 0.



(a) This figure shows a simple x-monotone path.



(b) It is illustrate how the x-coordinates are determined.



(c) The remaining x-coordinates are calculated.

Figure 5.22: This figure shows how to calculate the x-coordinates of all vertices in the schematized path. Given that each edge has an assigned angle and all actual horizontal strip heights have been determined.

Any other horizontal strip height can be 0 or larger. As we already showed, any horizontal strip height set to any value equal or larger than 0 cannot lead to a violation of the orthogonal order (see Lemma 5.3.3). We change the horizontal strip heights determined by Algorithm 2 so that any horizontal strip height which does not change the produced cost is set to 1. The process to do that is fairly easy. The idea is to check for any horizontal strip whether its height is 0 and if a h-edge would be pushed open if the horizontal strip height would change to 1. If that is not the case the horizontal strip height is set to 1. This procedure cannot lead to a higher minimum cost. It only introduces more options for the LP to determine the optimal solution. In Figure 5.23 one example is shown. On the left side is a polygonal path. On the right side the result of the schematization process is depicted. All edges have been assigned their preferred direction while maintaining the orthogonal order. However, the result is unfavorable. All vertices above $y_u(s_i)$ are placed at the same y-coordinate. And all vertices below $y_l(s_i)$ are set to the same y-coordinate. Depending on the configuration of the formulated LP the resulting path may become very large.

Algorithm 5: `JoinNodes`

```
Input: Precomputed list of horizontal strip heights and $x$-monotone path $P$
Output: Coordinates for each vertex of $P$.
$U = \text{sort}(P)$;
$n = |U|$
index = $\mu^-1(n)$;
$y(v_{index}) = 0$;
prev_index = index;
for $i \leftarrow n-1$ down to 1 do
    index = $\mu^-1(i)$;
    $y(v_{index}) = r(i) + y(v_{prev\_index})$;
    prev_index = index;
$x(v_1) = 0$;
for $i \leftarrow 2$ to $n$ do
    $x(v_i) = x(v_{i-1}) + \cos(\alpha) \cdot |y(v_i) - y(v_{i+1})|$;
```

The algorithms running time is in $O(n)$.

5.4 Level-1 and Level-5

Basic Principle. The vertices and edges of Level 1 are determined by a simple principle. The basic assumption about Level 1 is that right at the beginning the route takes numerous turns and the edges are very short. This leads to the conclusion that a schematized version of this part of the path may be unnecessary and may yield no better readability compared to the geographically correct version. The algorithm constructs a path beginning with the start vertex of the route. It travels along the path until one of the following conditions has been met:

1. The maximum category has been reached.
2. The maximum distance has been reached. Tests have shown that 15 km is a good maximum distance.
3. The path has reached a long street with no turning points.

All of these require that a certain minimum distance has been reached.

Enriching Information. The general idea behind the enrichment process is to add vertices and edges which help the user orientate himself and find the correct route. The edges are not altered however some vertices and edges are omitted. The result of this process is that all edges in the set generated are either part of the path or directly at an intersection or close to an intersection.

After the end vertex of the Level 1 subpath has been determined the algorithm visits each intersection on this subpath and starts a simple depth search first (DFS)(see [CLRS01] for an explanation). However, the DFS terminates if the edge has a category which is too low or the edge is too far away.

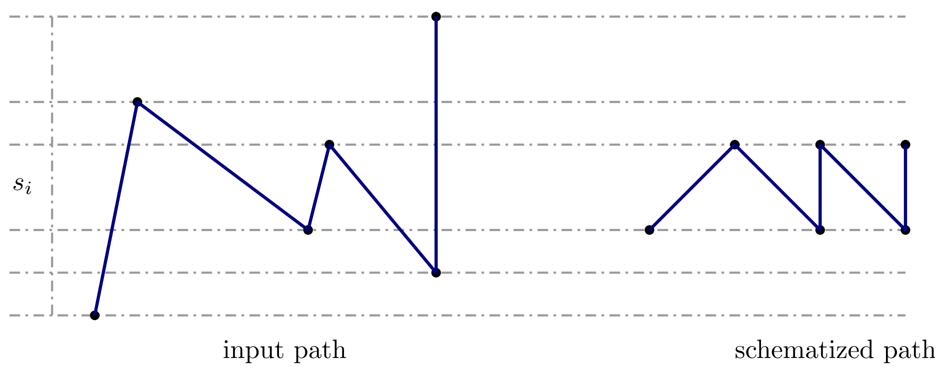


Figure 5.23: To the left is a polygonal x-monotone path. This path is to be simplified with our schematization algorithm. The result of the schematization process is depicted on the right (there are more than one possible valid solutions). Every edge has its preferred direction assigned. Only the horizontal strip s_i has been assigned the height of 1. All other horizontal strips have height 0.

5.5 Level-3

Finding the monotone subpath in the central path. As already explained in the introduction and based on the observation mentioned in [MHBB06] normally a route consists of 3-5 major parts. In most cases there is a long path on a highway that makes up a large part of the total route. In this section the goal is to describe how to extract this part. The idea is very simple and straightforward. First the algorithm calculates the total length of the path after Level 1 and 5 have been removed and the Douglas-Peucker algorithm (see Chapter 5.2) has been applied to it. The total length is then divided by two and the vertex nearest to that distance is chosen as `start vertex`. Beginning from this vertex the algorithm searches in both directions. It tests for a y- or x-monotony separately and stops if the next vertex would destroy the current monotonicity property or the street category reached is too low. After both y- and x-monotone subpaths have been extracted the longer is chosen. The schematization algorithm described in Chapter 5.3 is applied. The resulting path is still monotone and the orthogonal order is maintained. If the number of vertices of the given path is n, the algorithm which extracts the longest monotone subpath has a worst case running time of $O(n)$.

5.6 Level-2 and Level-4

Introduction. After removing the Level-3 path the central path is split into at most two subpaths. We denote them as Level-2 and Level-4.

Finding large monotone subpaths. It is possible that the path in Level-2 or -4 is not x- or y-monotone. Then, the algorithm acts greedily. It calculates the longest monotone subpath beginning from one end vertex of Level 3. If this path does not cover all vertices of Level-2, a second monotone subpath is determined beginning with the end vertex of the first subpath. This process is repeatedly applied until Level-2 has been split into a set of monotone subpaths. This is also applied to Level-4 beginning with the other end vertex of the Level-3 path.

Note that it may be the case that there are neither a Level-2 nor a Level-4. This can only happen if longest monotone subpath of Level-3 ends at end of Level 1 (or at the beginning of Level 5).

Schematizing. After the set of monotone subpaths has be determined the schematizing algorithm described in Chapter 5.3 is applied to each monotone subpath separately. The result is a set of schematized monotone subpaths. Note that the path has no unique scale. This is, of course, a side effect which is caused by the value of the minimum edge length. It is obvious that in Level-2 and -4 the edges are a lot shorter than in Level-3 where mainly only highways are being used. Whereas in Level-2 and Level-4 smaller streets will be part of the route leading to smaller edges. This may lead to problems where a Level-2 (or Level 4) monotone subpath may create a false intersection with another monotone subpath of Level-2 (or Level-4). Further, it is possible that false intersections with Level-3 are created. This may irritate the user and must be avoided (see Chapter 5.7).

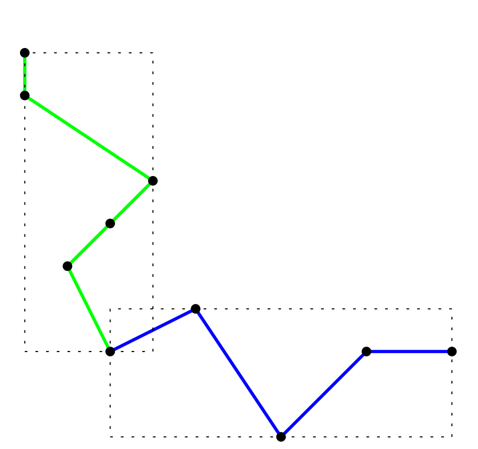
5.7 Combining all Parts

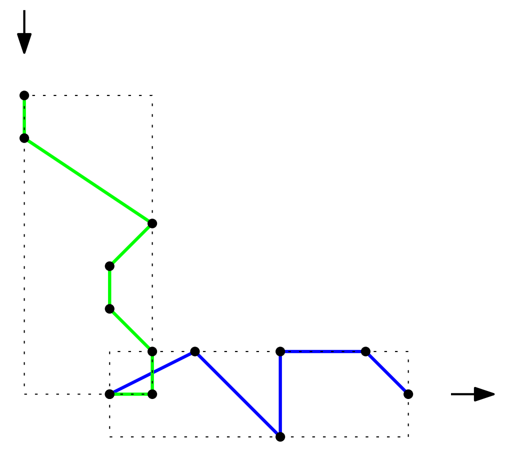
As mentioned in the previous sections we combine the schematized paths of all different levels. This results in several different and independent sub paths. Further, we changed the scale of these paths by introducing the minimum length and by restricting the directions to very few. Due to the way we discover the different Level-2 and Level-4 parts and the different scale it can happen that the different Level-2 paths overlap each other. This can lead to false intersections and may irritate the user. An example can be seen in Figures 5.24(b) and 5.24(c). We solve some conflicts by inserting small additional edges. If the schematized path overlap over Level 1 or Level 5 we move this part to a nearby location.

5.7.1 Avoiding False Intersection

It is undesirable to introduce intersection of path segments where in reality there none. We call these intersections *false intersections*. This can happen due to multiple reasons. We restrict the direction of all edges, we change the minimum length and we alter the scale of the path. Note that a false intersection can only occur between to separate monotone paths. It is impossible that a monotone path produces a false (or even any) intersection. For avoiding false intersections we actually consider the rectangles which enclose the monotone paths. However, to detect we test if line segments overlap each other.

Two sequential monotone paths. A simple example is shown in figure 5.24(b). There, we can see two monotone paths which are connected by one vertex. The blue path is y-monotone and the green path is x-monotone. The intersection between the green and the blue line segment is a false intersection. To solve this problem we introduce an additional edge between two sequential monotone paths. This edge has length 0 and its direction is the same as the direction of the path (i. e.,if it is an x-monotone path from left to right the direction is horizontal to the right, if its a y-monotone path from bottom to top the direction is vertical from bottom to top, etc.) or the direction is orthogonal to that of the path. We solve the problem by increasing the length of this edge. To simplify things we increase the length of the edge so that the rectangles that enclose the monotone paths do not overlap each other. This is shown in figure 5.24(c). The edge which is highlighted in red is the newly


(a) An example of two subsequent not conflicting subpaths.


(b) An example of two subsequent conflicting subpaths

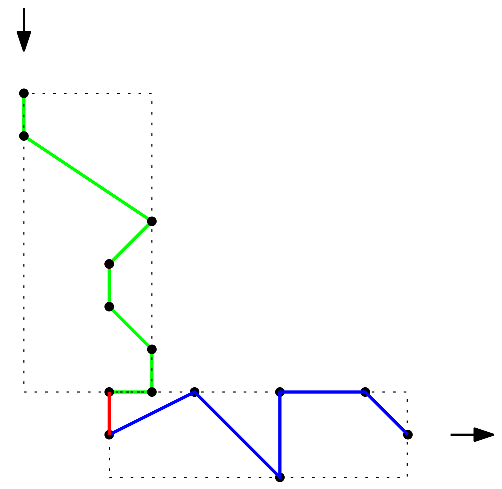

(c) Illustration how to resolve conflicts by adding one edge (highlighted in red).

Figure 5.24: Conflict resolving of two subsequent monotone subpaths.

introduced edge which with initial length 0. We increased the length so that the rectangles do not overlap. In the following we only consider the enclosing rectangles for our problem and not individual paths.

First, we prove that a monotone path can only have an intersection with another path.

Lemma 5.7.1

Any x- or y-monotone path does not self intersect.

Proof: Let P be a monotone path. Without loss of generality we assume its an x-monotone path from left to right (the general idea of this proof is for all monotone paths the same). Thus, for every $v_i \in P$ the following holds:

$$\forall v_i, v_j \in P, i < j : x(v_i) \leq x(v_j)$$

An intersection would require that there is at least one $v_i \in P$ with a smaller x-coordinate than its left neighbor (i. e.,$x(v_i) < x(v_{i-1})$). Even if all x-coordinates of all $v_i \in P$ are the same this only yields a straight vertical line. No intersections are possible (this would be of course a y-monotone path). If there is a $v_i \in P$ with $x(v_i) < x(v_{i-1})$ it would follow that P is not x-monotone from left to right. ■

Now, we prove that two sequential monotone paths cannot produce an intersection.

Lemma 5.7.2

Two different monotone paths can only have an intersection if their bounding boxes overlap.

Proof: The paths are within their enclosing rectangles. If the enclosing rectangles do not overlap each other it is impossible that any two edges create an intersection. ■

Finally, we prove that two sequential monotone paths can be made conflict free if we insert a new edge of certain length.

Lemma 5.7.3

We can connect two monotone paths of different orientation by inserting between them an additional edge of a certain length.

Proof: As already proven in lemma 5.7.1 we know that two sequential monotone paths can only have an intersection if they have a different monotony. Thus, we only consider this case. We can choose to resolve the conflict (i. e. the intersection) with two different edges.

Let P_1, P_2 be two paths. Without loss of generality we assume that P_1 is an x-monotone path from left to right and P_2 is an y-monotone path. Further, let the last vertex of P_1 be equal to the first vertex of P_2 and if we draw the vertices of both paths and connect them via edges the resulting drawing shows at least one intersection. As we already know this intersection is caused by an edge of P_1 and an edge of P_2 (see lemma 5.7.1).

Case 1: The first edge in P_2 has a non horizontal direction, or it is a horizontal edge but the orientation of edge is from left to right and not from right to left.

We resolve the conflict by adding a dummy edge between the two monotone paths. This edge is horizontal and its orientation is from left to right. The length of the edge is chosen so that the rectangles which enclose both paths do not overlap. As proven in 5.7.2 there can be no further intersection caused by these two paths.

Case 2: The first edge in P_2 is a horizontal edge with orientation from right to left.

We cannot solve this problem the same way as we in case 1, because we would introduce an new intersection that is caused by the dummy edge and the first edge of P_2. Both edges would be drawn on top of each other. Thus, we introduce another dummy edge with a different orientation. To determine which orientation we determine the first y-monotone path P_y after P_1. Note, this can be P_2 but it can not be guaranteed. We determine the general orientation of P_x (i. e.,top to bottom or bottom to top). The new dummy edge gets this orientation assigned. The length is chosen so that the two rectangles that enclose P_1 and P_2 do not overlap.

At least one case is always possible. Sometimes both cases can be possible and one might chose the case which yields a smaller dummy edge. We have shown a technique to connect two monotone paths so that they do not create an intersection. The proof for a different combination of monotone paths follows analogously. ■

Multiple paths. We have shown in Lemma 5.7.3 that we can connect two consecutive monotone paths so that there are no intersections. However, as there are potentially many monotone paths that we wish to connect there is a possibility that two non-adjacent monotone paths cause an intersection. An example is shown in Figure 5.24(b), where the insertion of a y-monotone. In this figure we only show the enclosing rectangles and not the monotone paths themselves. We illustrate a technique to resolve the conflict (i. e.,intersection). See Figure 5.25 for an illustration. When we append monotone paths (with the technique shown above to resolve immediate conflicts) and encounter a conflict we calculate how much the newly appended monotone path has to be moved in y-direction or x-direction to resolve the conflict. Our description deals with vertically resolving conflicts. The same technique works horizontally in an analogous way. In Figure 5.25 there is a horizontal dashed blue line which indicates where the appended monotone path has to end so that there is no conflict. After we know this line we determine the edges which are intersect the line. Note, there has to be at least one edge the line that is intersected. Otherwise the conflicting monotone paths would not be connected. Further, we do not consider edges of the newly appended monotone path. These edges remain untouched.

There are three cases to distinguish:

1. The line intersects an edge of a y-monotone path. The direction of the edge has a vertical component.
2. The line intersect one (or more) consecutive edge(s) of a y-monotone path. The direction of the edge(s) is horizontal.
3. The line intersects one or more edges of an x-monotone path.

In the following we present a technique which acts for every case differently. The result is that the conflict has been resolved and the last appended monotone path does not produce a conflict (and no new conflict has been introduced).

Case 1: We know the line intersects an edge of a y-monotone path. We know that this edge is non horizontal. This means we can create two new dummy vertices along this edge and add between those vertices a new edge. This edge will be vertical and the length of this edge will be the y-distance we calculated earlier to resolve the conflict.

Case 2: We again introduce two new vertices and an edge. They are added to one of the edges which are intersected by the line. The new edge connects the two vertices and is vertical (as in case 1). The length of this edge equals the earlier calculated y-distance that is enough to resolve the conflict. However, special consideration has to be taken so that the orthogonal order inside this monotone path is not violated. The schematized path has set both vertices to the same y-coordinate. We can not choose freely which vertex is moved down or up. The orthogonal order is maintained if we follow the path in its direction and move the vertex which is later in the path by the appropriate y-distance.

Case 3: Suppose, the x-monotone path P_k, which is intersected by the line, is not the first monotone subpath of P. This implies that there is another monotone subpath preceding P_k. Between this previous path P_{k-1} and P_k there might be a dummy edge (see the problem of combining two sequential monotone paths). If there is no vertical dummy edge we add a vertical dummy edge. Otherwise we pick the existing vertical dummy edge. We add to the

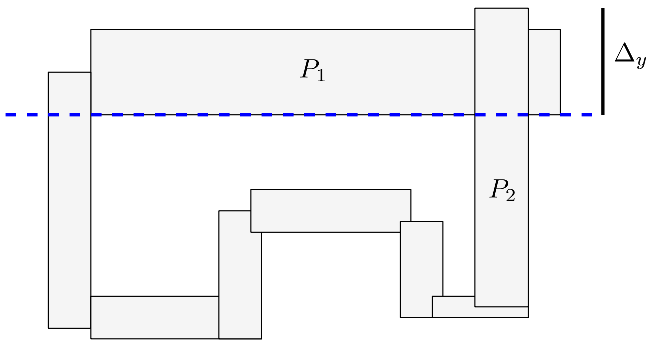


Figure 5.25: This figure shows the enclosing rectangles of multiple monotone paths. The path P_1 and P_2 produce a conflict. We can solve it by moving the path P_2 by Δ_y down.

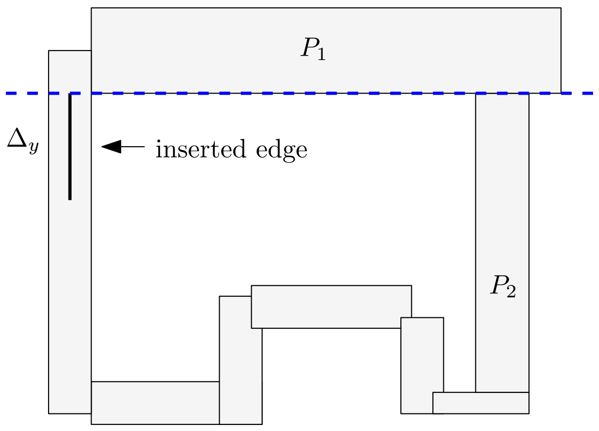


Figure 5.26: Here, we inserted an edge of length Δ_y into the path which is intersected by the dashed, blue line. Thus, moving all paths down. The conflict between P_1 and P_2 has been resolved.

length of the dummy edge the y-difference we need to resolve the conflict. Thus, we "push" the whole path below or above the line.

If the x-monotone path is the first monotone path P_1 of P this means both conflicting paths are appended after the this path. Thus, changing the position of this path does not change the relative positions of the conflicting paths and we cannot resolve the conflict. However, to avoid possible conflicts we have to push this path down by the y-difference.

If we know how much x-distance is needed to resolve the conflict we can apply the basically same technique as above. An example is shown in figure 5.27 and figure 5.28. Note that it is possible that both changing the x- or the y-distance leads to resolving the conflict (see figure 5.29). Resolving the conflicts by moving the monotone subpaths in y-direction is always possible. We move all monotone subpaths below the dashed, blue line by the same amount of Δ_y downward. However, resolving conflicts by moving the monotone subpaths in x-direction does not necessarily resolve the conflicts. The problem here is that we push the monotone

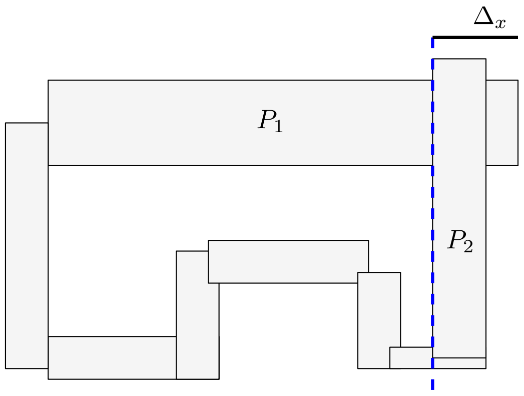


Figure 5.27: This figure shows the same conflict as shown in figure 5.25. However, this time it is shown how to solve the conflict by moving the path P_2 to the right.

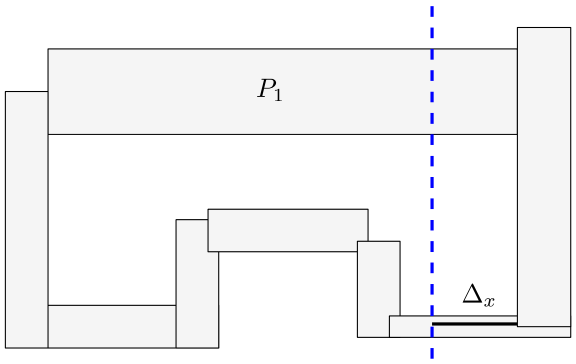


Figure 5.28: The conflict has been resolved. An edge of length Δ_x has been inserted into one path. This pushed the path P_2 to the right.

subpaths in one direction until the conflict we wished to solve has been resolved. However, we do not necessarily move all monotone subpaths in this direction so that no new conflict can occur. As can be seen in Figure 5.30. But this is easily detectable. We only need to ensure that the position we want to move the new appended monotone subpath is not already occupied by another monotone subpath which itself is not moved by our technique (i. e.,there are no monotone subpaths preceding it that are intersected by the dashed, blue line).

Lemma 5.7.4

The technique described above is able to resolve conflicts of two non-sequential monotone paths. Further, it introduces no new conflicts.

Proof: The basic idea is to increase the length of certain edges so that the newly appended path does not produce a conflict. First, we prove that the conflict will be resolved by this method. Second, we prove that there can be no new conflicts caused by this method.

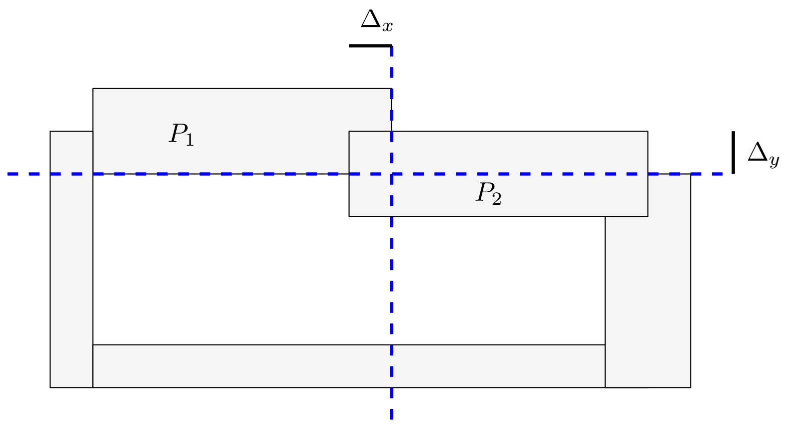


Figure 5.29: This figure illustrates another example where the conflict can either be solved by moving one of the conflicting paths by a Δ_y or a Δ_x.

Conflict resolving. We restrict our proof to the case shown in figure 5.25. The proof for the remaining cases follows analogously. Let there be a conflict between two monotone paths P_1 and P_2 which are not directly connected. Let P_1 be an x-monotone path and P_2 a newly appended y-monotone path (see Figure 5.25). Let $\min_y(P_1)$ be the smallest y-coordinate among all points of P_1. Let $\max_y(P_2)$ be the largest y-coordinate among all points of P_2. If P_1 is pushed downward in y direction by the difference $\Delta_y = \max_y(P_2) - \min_y(P_1)$ the bounding boxes of P_1 and P_2 do not overlap. Thus, the conflict has been resolved.

We create a line l which is parallel to the x-axis and has the y-coordinate $\min_y(P_1)$. This line intersects at least one monotone path P_i strictly between P_1 and P_2 (otherwise P_1 and P_2 would not be connected). In every path P_i which is intersected by the line we insert a new edge into the edge which is intersected. The mechanism has been described above. The length of this new edge is Δ_y. If P_i is y-monotone, the path has increased its height by Δ_y. By increasing this height, P_2 gets pushed downward by Δ_y since its position depends on the coordinates of the last node p_i. Suppose, there is no such P_i but only x-monotone paths which are between P_1 and P_2. Then because of the increased length of the dummy edge (see case 3), the path P_2 is again pushed down by Δ_y. This is enough to resolve the conflict because the enclosing rectangles cannot overlap.

No new conflicts. We have to prove that this technique does not introduce new conflicts. Conflicts could be possible if the enclosing rectangles of paths would overlap after we have resolved a conflict. In cases 1 and 2 we described how to increase the height of y-monotone paths. This was done by inserting a vertical edge. This does not increase the width of the path. Because we changed all y-monotone paths and all x-monotone paths which are intersected be the line so that they have all either an increased height or are pushed down by Δ_y there cannot be an new conflict.

We could show that we could resolve conflicts of two monotone paths and we do not introduce new conflicts. Thus, we presented a method to append monotone paths without any false intersections. ■

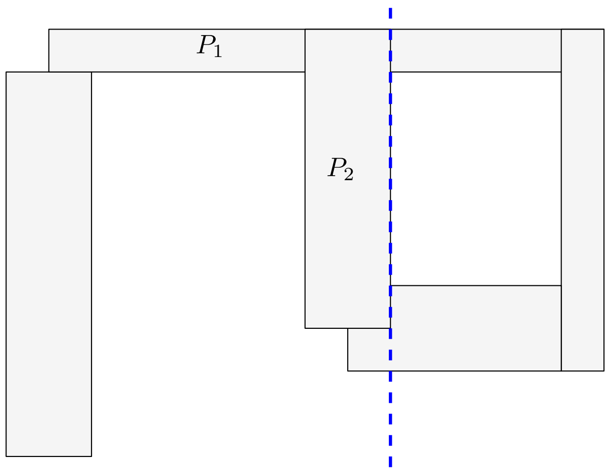


Figure 5.30: This figure illustrates that it is not always possible to resolve the conflict by moving one of the conflicting paths by an appropriate Δ_x. Here, moving P_2 to the left cannot resolve the conflict.

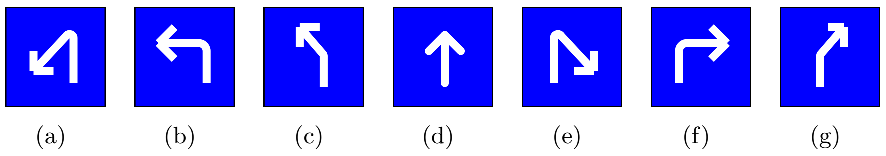


Figure 5.31: The shown figures are the street signs which indicate the turn direction at intersections.

5.8 Decoration

This section explains how additional decoration is placed to enrich the information provided by the route sketch.

5.8.1 Street Signs

It was already explained we use several labels to enrich the information given to the user. We use seven different signs to indicate the change in direction at intersections. At some intersections these signs are omitted. For example, if the user has to follow a street and simply has to drive through an intersection. The seven different signs are shown in 5.8.1 and are based on the wayfinding choremes defined by Klippel [Kli03]. These seven directions proved to be sufficient for the participants of a study to describe the change of direction one has to take at an intersection.

Further, we use a sign for roundabouts. On this sign all different exit streets of the roundabout are shown. They are not aligned to the restricted directions of the main algorithm, but rather are based on the original directions of the road network. The entry and exit road are highlighted on the sign to help the user find the right exit (see Figure reffig:streetsign1).

We use one street sign indicating the beginning of a highway and one to indicate the end of a highway. If the route changes from one highway to another it is indicated by another

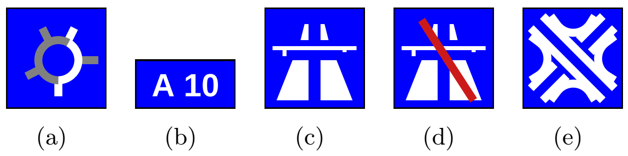


Figure 5.32: The five figures above show some street signs used in our route sketch design algorithm. Figure (a) indicates a roundabout, (b) shows a highway name, (c) denotes the begin of a highway, (d) denotes the end of a highway, and (e) indicates a highway intersection.

sign (see Figures 5.32(c) – 5.32(e))

Finally, we use signs to indicate highway names (e. g., “A 10”, see Figure 5.32(b)) the user has to take. They are simply placed on the highway itself.

Overlapping Street Signs. It is undesirable that two signs overlap each other or that a sign overlaps other a road. This is to be avoided. One typical variant of the label placement problem is as follows; Given are n points. The goal is to assigning each point one axis-aligned rectangle in a way that the point is located on one corner of the rectangles. All rectangles have the same size. The size of the rectangles is to be maximized. In [FW91] Forman and Wagner could show that this problem is NP complete. However, our problem is slightly different. We align the signs to the streets of the graph. This effectively means that we might have to deal with rotated rectangles.

The algorithm we use is based on the algorithms introduced in [Yam07] which themselves are partly based on [SVA00]. Further, we have two possible positions for the street sign. One on the right and one on the left side of the street. The right side is the preferred side meaning, if both sides are possible the algorithm will choose the right side. The sign is placed with a certain distance to the road. All signs have the same size and the objective is *not* to find the largest possible size but a solution that minimizes the number of overlaps.

The Conflict Graph. The first thing that the algorithm does is to construct a conflict graph. First, we determine the possible positions for all street signs. If one position will overlap a street the position is not considered at all. After all possible positions for the street signs have been determined we test for conflicts among them. A conflict occurs if any two shapes overlap each other. If our shapes would be axis-aligned rectangles of equal size the calculation of conflicts would be very easy. We simply would test if any corner point of a rectangle is “inside” another rectangle. Let the number of possible positions be n. The algorithm would take in a naive form $O(n^2)$ time.

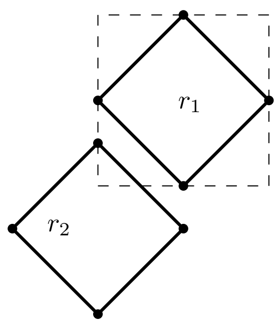
Unfortunately the rectangles are not necessarily axis-aligned but they are of equal size. Determining if a conflict occurs is not very hard. For any two rotated rectangles we can make a rough assessment if they are even able to produce a conflict. For that we determine the minimum and maximum x- and y-coordinates of one rectangle. If no corner point of the other rectangle lies between the minimum and maximum x- and y-coordinates a conflict cannot happen. Note, this can only be guaranteed if all signs have the same size and shape. The former assessment is only a necessary condition for a conflict but not a sufficient one (see Figure 5.33(a)). So for those pairs we can test if a conflict actually occurs if we consider the edges of both rotated rectangles. If any two edges cross each other a conflict between those two rectangles has been shown (see Figure 5.33(b)). Again we are able to utilize range trees. This time in the more traditional sense. We insert all corner vertices of all shapes into a range tree. After this is done we can extract for any rectangle the vertices (and thus the candidates) for a conflict. Let n be the number of rectangles. The construction of the range tree can be done in $O(n \log n)$ time. The extraction of the points can be done in $O(\log n{+}k)$ where k is the number of extracted points. We have to test n different rectangles for a conflict. Determining the candidates for conflicts can be done in $O(\log n + k)$. Testing for crossing of two edges can be done in $O(1)$ time. The total running time is $O(n \log n) + O(n) \cdot (O(\log n + k) \cdot O(1))$. This means that the total running time is not dominated by the construction of but by the determination of the conflicts of each rectangle. In a worst case scenario during each extraction of the possible candidates for a conflict we get all n possible rectangles. This would mean the total running time is in $O(n \log n + k)$, where k is the total number of conflicts.

For every rectangle we store the number of conflicts it produces and with which rectangles these conflicts appear and which point the conflict belongs two (recall, we only need one position for each point). The idea is to reformulate this problem into `Maximum Independent Set`. This problem is NP-hard [GJ79]. We use a greedy heuristic that always selects the label position with the fewest conflicts. This technique enables us to create a maximum-inclusion independent set [SVA00].

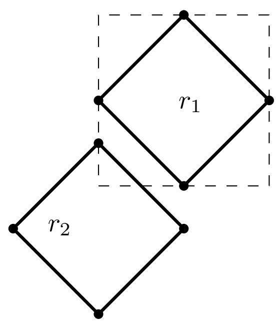
We use this information to build a conflict graph. One vertex for each label position and an edge for each conflict and between any pairs of rectangles for the same point. The goal now is to find a maximum independent set in this graph. This corresponds to a conflict-free labelling with a maximum number of labels. An entry with value 1 indicates a conflict between each rectangles. An entry with value 0 indicates no conflict. A rectangle cannot produce conflict with itself (i. e. the diagonal from the upper left to the lower right of the matrix is filled with entries of value 0). The rectangles are now vertices inside the conflict graph. We determine the vertex with the smallest degree (i. e.,the least conflicts). The vertex and all adjacent vertices to it are removed (i. e.,all rectangles which produce a conflict with the chosen rectangle). If there is a vertex left belonging to the point it is removed too. The chosen rectangle for the point is saved and the process is repeated. Unfortunately, the illustrated process may lead to several points without any assigned rectangle for the street sign. For this case we choose the preferred rectangle and accept that this leads to a street sign that overlaps another street sign.

To implement this algorithm we use a binary heap (see [CLRS01] or [ASSS86] for an introduction). In this binary heap we store each vertex. The key in this binary heap is the degree of each vertex. Given that we initially know the degree of each vertex (i. e.,the number of conflicts) the binary heap can be constructed in $O(n \log n)$ where n is the number of vertices in the conflict graph (which is the same as the number of rectangles). To ensure that the right side of the street is a favorable position for the street sign we add a small constant $\epsilon \in (0, 1)$ to the key corresponding the position to the left side of the road.

Each `extractMin` needs $O(\log n)$ time. During each `extractMin` operation we have have to do at most n `updateKey` operation which yield time of $O(n \log n)$. We have to do at most n `extractMin` operations which need time of $n \cdot O(n \log n) = O(n^2 \log n)$. Thus, the total running time of the algorithm is not dominated by the part which calculates the conflicts. The initial conflicts can be determined in $O(n \log n + k)$ time but the total running time remains at $O(n^2 \log n)$. Algorithm 6 summarizes the above.



(a) No conflict.



(b) A conflict occurs

Figure 5.33: Depicted are two rotated rectangles. In Subfigure (a) the rectangle r_2 satisfies the necessary conditions for a conflict. However, the algorithm determines that there is no conflict. In Subfigure (b) a conflict is shown.

Algorithm 6: Maximum non conflict labelling algorithm

```
1  Construct conflict graph;
2  Create Binary Heap $b$;
3  Insert all vertices with their degree as key into $b$;
4  Create set $S := \emptyset$;
5  while $b$ not empty do
6      $n = b$.extractMin();
7      insert $n$ into $S$;
8      determine the point $p$ the vertex $n$ belongs to;
9      remove all vertex conflicting with $n$ from $b$;
10     remove all vertices belonging to the same point as $n$;
11     call the updateKey() function for all vertices conflicting with vertices belonging
       to $p$;
12 return $S$;
```

5.8.2 Magnifier

We presented a technique to eliminate false intersections of monotone subpaths. However, we chose to deal with conflicts of Level-2 and Level-4 and either Level 1 or Level 5 differentially. Recall that Level 1 and Level 5 are both not schematized. They remain in their original representation. Hence, the technique provided (inserting a dummy edge) is not suitable.

Our idea is to move the Level 1 or Level 5 area (the shape is a rectangle) to another, close location. Of course the new location must not introduce new intersections. There are some ideas on how to maintain the free space. That is the area of the picture where nothing is drawn. An interesting proposition is explained in [BJ97]. In their paper Bernard and Jacquenet explain a method on how to calculate the free space of an area when rectangles are inserted into it. The general idea is the following:

- Maintain a set *LFRSet* which contains the largest free rectangles.
- Maintain sets *NewSet* and *RemoveSet*. Initialize both of them with $\emptyset$.
- Initialize LFRSet with the rectangle that spans the drawing area.

- After inserting a rectangle. Look at all sides of this rectangle and determine the rectangles of LFRSet which are affected by the new rectangle. Insert those rectangles into RemoveSet. Determine how those rectangles are affected by the new rectangle and calculate new free rectangles. Insert those new rectangles into NewSet.
- LFRSet = LFRSet $\setminus$ RemoveSet
- LFRSet = LFRSet $\cup$ NewSet

After each iteration LFRSet contains the largest free rectangles.

Unfortunately, our problem differs a bit from the description above. As we do not simply insert rectangles. We have a polygonal path and street signs (rectangles which are potentially rotated by any degree contained in $\mathcal{C}$). A simple idea is to subdivide the drawing area into squares. To maintain the street signs we simply insert a rectangle for each sign which completely encloses this sign, similarly to the dashed square in figure 5.33(a). Inserting the path is a little bit more tricky. We could simply insert rectangles which enclose line segments. However, this may block areas that are too large (see figure 5.34). Thus, we use an underlying grid of squares with a fixed size. Then, for every segment of the path we determine the squares of the grid that are intersected by the segments. Those squares are used as blocked rectangles in the algorithm of Bernard and Jacquenet. This significantly reduces the blocked area. Although this approach does not necessarily yield optimal results (i. e.,all free pixels) this not necessary. The free area we "lose" is very close to the path or the street signs. This area is not a good candidate anyway because the route becomes too cluttered if everything is drawn very close to each other.

If any part of the central path intersects the drawings of Level-1 or Level-5 we determine an area that is close to the original position and included in the free space. This is where we draw the Level-1 or Level-5 rectangle. We indicate its original position by connector lines and a small rectangle.

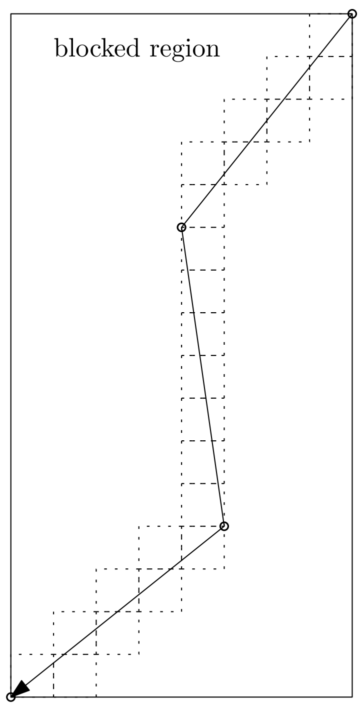


Figure 5.34: The figure shows a path and its enclosing rectangle. However, if we would use this rectangle to mark an area as blocked we would waste much space. It is much more useful to use only the small dotted squares to block regions for drawing.

Chapter 6

Experiments

Introduction. In the following we will present experimental results of our schematization and route map design algorithm. The first section is devoted to analyzing measurable criteria of our approaches. These results helps us to evaluate certain assumptions we made in the previous sections. Although our goal is mainly an aesthetic one we still want to determine if our algorithm has a reasonable running time. This is discussed in the later part of this section. In the section section, we present case studies of certain examples and show how changing some parameters of our algorithm change the visual representation. Finally, we present some examples.

All results are based on the road network of Germany consisting of 5,137,911 and 11,184,562 edges. The data these findings are based upon is kindly provided by the PTV AG[1]. All experiments are conducted on an Intel(R) Xeon(R) E5430 CPU clocked at 2.66GHz with 32 GB of main memory. The program was written in C++ and compiled with GCC 4.3.1. and optimization parameter `-O3`.

6.1 Experimental Evaluation

All following results in this section are based on 10,000 randomly chosen origin and destination vertices from the road network of Germany. The shortest paths were computed with Dijkstras algorithm [Dij59]. If not explicitly stated otherwise the value for ϵ is 100,000, the value for the minimum length is 150,000 and the set $\mathcal{C}_{30} = \{0^\circ, 30^\circ, \ldots, 360^\circ\}$ ist used as set of allowed directions.

Prefix and Suffix Path. The first part of our algorithm extracts a pre- and a suffix path. We introduced certain criteria how many vertices can be part of these paths. This is explained in Chapter 5.4. Table 6.1 shows the percentage the pre- and suffix make up make of the total path length. As we can see they tend to be very small. In fact, there is no pre- or suffix path that makes up for even 4% of the road in all tested examples. Thus, the chosen parameters for determining the pre- and suffix path lead to very small pre- and suffix paths.

[1](see http://www.ptvag.com/)

Table 6.1: This table shows the length of the pre- and suffix path of the total path. The vast majority of all examples make up for less than 0.2% of the total path length.

Total length [%]	[0, 0.1)	[0.1, 0.2)	[0.2, 0.3)	[0.3, 0.4)	[0.4, 0.5)	> 0.5
Prefix Path – Frequency	70.59%	22.59%	4.29%	1.57%	0.45%	0.51%
Suffix Path – Frequency	39.00 %	39.02 %	12.53 %	4.74 %	2.05 %	2.65 %

Table 6.2: This table depicts the length of the Level-3 path in relation to the total path length.

Length of Level-3 Path [%]	[0, 20]	(20, 40]	(40, 60]	(60, 80]	(80, 100]
Frequency	15.05%	27.41%	24.97%	19.28%	13.29%

6.1.1 Central Path

Douglas-Peucker Algorithm. As the Douglas-Peucker algorithm plays an important role in our algorithm we discuss how many vertices the algorithm removes. Keep in mind that we have to use a modified version of the Douglas-Peucker algorithm to ensure consistent turn directions (c.f. Chapter 5.2 for an explanation). As explained, the parameter ϵ influences the number of vertices removed. In Figure 6.2 we show how many vertices are removed by the Douglas-Peucker algorithm depending on the ϵ parameter. The whisker plot shows that there is a noticeable difference between the results for the first four values for ϵ. However, after the value of ϵ has been increased to 100,000 only a very small increase in removed vertices can be seen. The mean number of removed nodes is for every tested value of ϵ always above 85% of all nodes. The Douglas-Peucker algorithm in its pure form would at some point remove all but two nodes of every path. However, we placed certain restriction as to which vertices must not be removed. This is why the percentage of removed nodes cannot grow beyond a certain threshold as the value for ϵ is increased.

Figure 6.1 shows how many vertices the Douglas-Peucker algorithm is able to remove. In the vast majority of tested examples, the algorithm removes more than 80% of all vertices.

Level-3 Path. The main part of our approach is the schematization algorithm developed in this thesis. It determines the height of the horizontal strips and thus the position of all vertices of the new schematized path. The algorithm is applied to Level-3 path. First, we discuss some facts about the central part of the path. In Table 6.2 we report the average length of the Level-3 path. This is, of course, done in respect to the length of the total path. We can see an almost even distribution for the length of the Level-3 path. It is conceivable that a Level-3 path is shorter (in relaion to the total length) if the total length is large. However, if the total length of the central path is short, the Level-3 path is generally longer compared to the total path length. Recall that the experiments are based on randomly selected origin and destination vertices in the road network. Hence, there should be an even distribution of long and short paths and this explains the even distribution of Leve-3 lengths.

Level-2 and Level-4. After the algorithm has determined Level-1, Level-5, and Level-3 and has schematized Level-3 the remaining part of the path is divided into several different

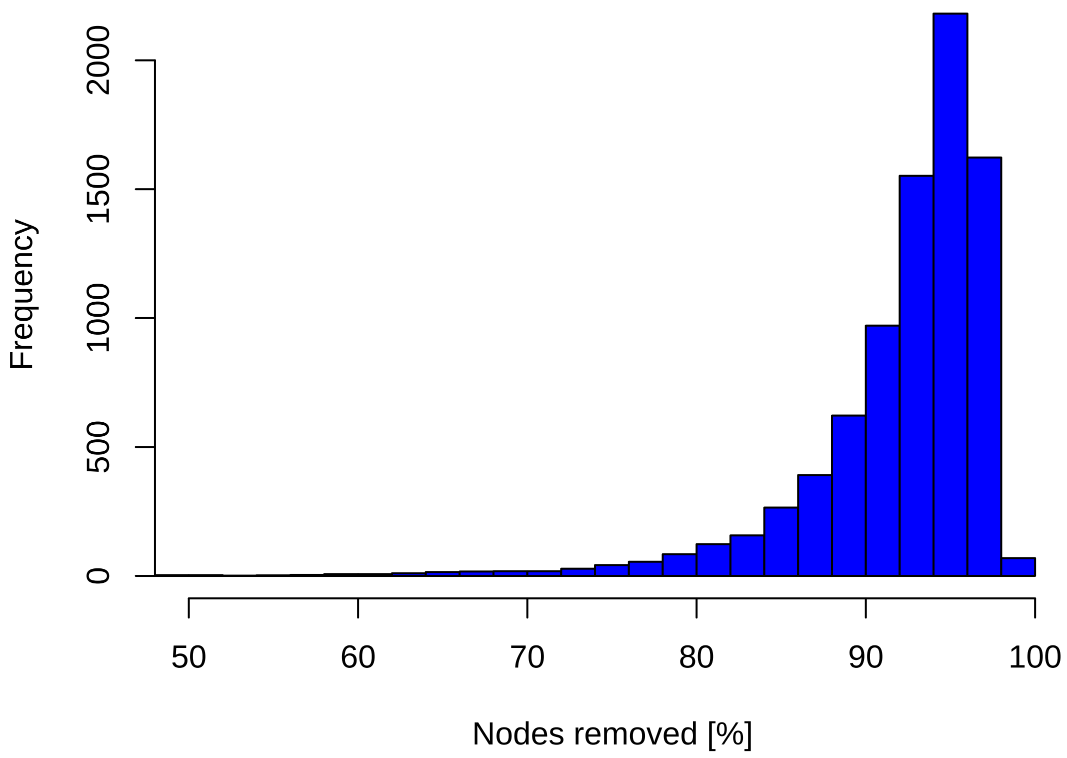


Figure 6.1: This figure illustrates how many vertices are removed by Douglas-Peucker algorithm. We set the ϵ parameter to a value of 15,000. For the vast majority of the tested routes, the algorithm removes more than 80% of all vertices of the path.

monotone subpaths. In Table 6.1.1 we show the average number of monotone paths of Level-2 and Level-4. The length of Level-2 and Level-4 related to the total path length is shown in Figures 6.3. It is noticeable that the path length of Level-2 and Level-4 is generally very short. This is not unexpected because the main part is already covered by Level-3. If, however, the central path is long, the Level-2 and Level-4 paths make up for more of the total path length.

Finally, we show in Table 6.4 the minimum cost of the schematization of each monotone sub path of Level-2 and Level-4. Although the monotone subpaths of Level-2 and Level-4 are generally shorter compared to the Level-3 path they produce (in average) higher cost. This is due to the different geographical aspects of many Level-2 or Level-4 paths. On a highway sudden turns occur rarely. In Level-2 and Level-4 paths these are more likely. This makes conflicts which are caused by vertex of the path that shares a v-edge and an h-edge more likely. However, the total cost for all monotone subpaths in Level-2 or Level-4 is low, i. e., in most cases at most 5.

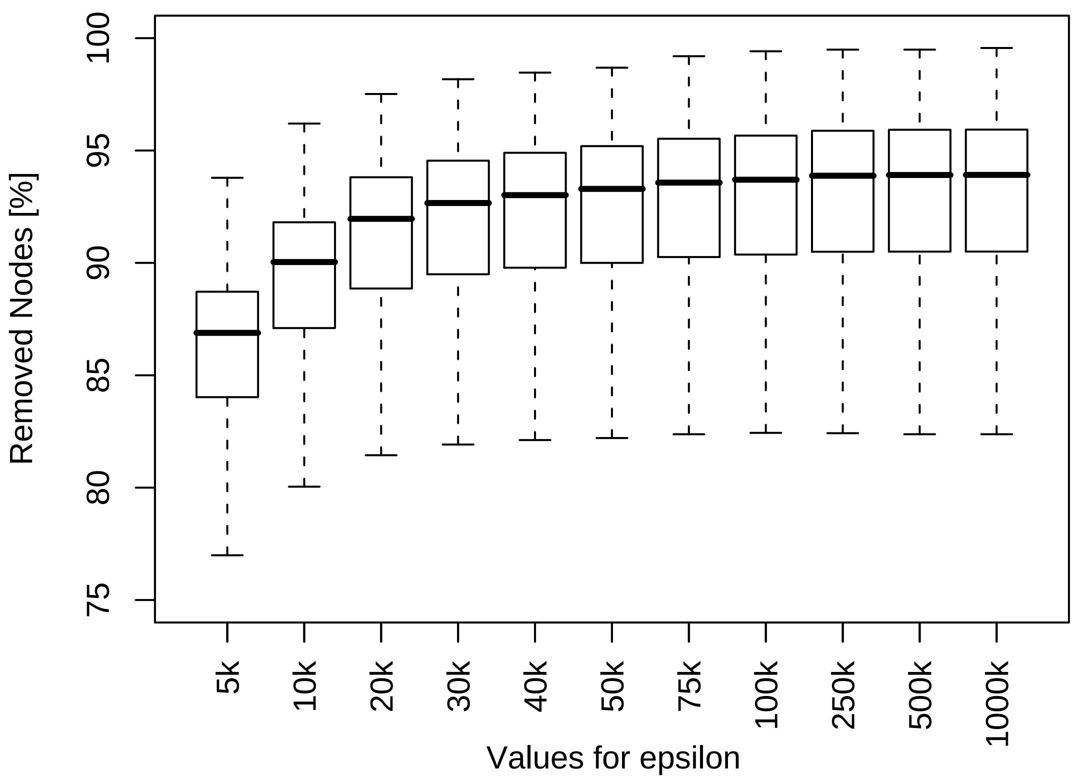


Figure 6.2: This figure illustrates the percentage of removed vertices by the Douglas-Peucker algorithm for different values for ϵ.

6.1.2 Schematization Algorithm

In Table 6.4 the minimum cost determined by the algorithm is displayed for Level-2, -3 and -4. We can see that the minimum cost is in most cases very low, i. e., 5 or less. For about every second example for Level-2, Level-3 and Level-4 path the minimum cost is 0. These results are encouraging. Only very few edges cannot be assigned their preferred direction.

Orthogonal Order. Another part we place particular concern is the orthogonal order (c.f. Definition 4.0.1). The orthogonal order in a schematized monotone subpath is maintained by our algorithm. However, because we change the minimum length and the directions of the paths in Level-2 and Level-4, we cannot guarantee that the orthogonal order between the different Levels is respected. We can also not guarantee that the orthogonal order is maintained inside Level-2 or Level-4, respectively. In Table 6.5 we illustrate how many pairs of vertices violate the orthogonal order after our algorithm has schematized all parts of the path. Because the number of vertices varies between different paths, we measure the number

Table 6.3: This table shows the number of monotone subpaths in Level-2 and Level-4. In about two thirds of all examples the number of monotone subpaths inside Level-2 and Level-4 does not exceed 4. Only in a very few cases the number of monotone sub paths is larger than 10.

# monotone paths	0	1-2	3-4	5-6	7-8	9-10	> 10
Frequency	20.05%	47.94%	21.87%	7.24%	2.06 %	0.58%	0.27%
Frequency	12.36%	48.83 %	27.14 %	8.28 %	2.09 %	0.81%	0.49%

Table 6.4: This table depicts the relative frequency of minimum cost determined by the schematization algorithm.

Minimum Cost	0	1	2	3	4	5	> 5
Level-2	57.44 %	24.55 %	10.51 %	4.07 %	1.93 %	0.87 %	0.64%
Level-3	53.32 %	28.78 %	10.23 %	4.48 %	1.71 %	0.66 %	0.82%
Level-4	56.76 %	24.90 %	10.47 %	4.29 %	1.72 %	0.98 %	0.88%

of orthogonal order violations in relation to the total number of vertex pairs. Although we cannot guarantee that the orthogonal order between the monotone subpaths is maintained, in about half of all tested paths the orthogonal order is violated only by very few vertex pairs. In most cases less than 25% of all vertex pairs are violating the orthognal order and in very few cases more than 50% of all vertex pairs violate the orthogonal order. The fact that violations of the orthogonal order do not occur very often is partly based on the fact that within one schematized monotone path there cannot be any violations due to the algorithm.

6.1.3 Combining the Monotone Subpaths

Resolving Conflicts. Because we restrict the edge direction and enforce a certain minimum length on the edges the we cannot ensure that different parts of the schematized path do not overlap. In Chapter 5.7.1 we explained how conflicts between the different schematized subpaths can be resolved. However, we show in Table 6.6 how often this is necessary. Although, theoretically this is poses a quite complex problem, the results presented here show that these conflicts in generally only very few schematized routes. Further, the experiments indicate that of all conflicts that arose, about 35,9 % of are between two monotone paths that follow each other. Resolving these conflicts by adding an additional edge as explained in Chapter 5.7.1 is easy.

Unfortunately, there are conflicts between the central and paths of Level-2 and 4 with the prefix and suffix path. These conflicts are fairly common. The prefix path has conflicts in 47.6% of all cases. The results for the suffix path are very similar. In about 52.2% of all paths tested a conflict arose.

Table 6.5: This table reports how many vertex pairs violate the orthogonal order. Considered are all vertices from Level-2, -3, and -4.

vertex pairs violating o.o.	0%	(0%, 5%)	[5%, 10%)	[10%, 25%)	[25%, 50%)	[50%, 100%]
Frequency	3.87 %	48.24 %	21.25 %	19.17 %	7.35 %	0.12 %

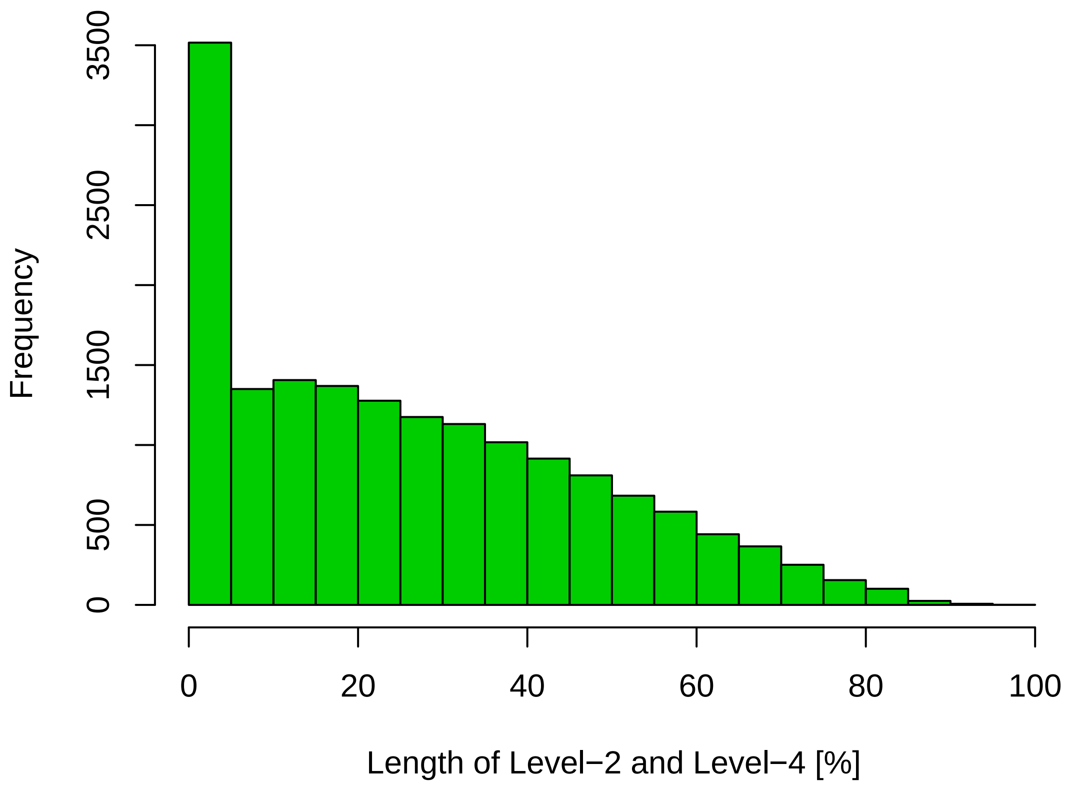


Figure 6.3: This figure reports the length of all monotone subpaths of Level-2 and Level-4. The length is given as percentage of the whole central path.

6.1.4 Decoration

Street Signs. The last step of our algorithm places decorations on the path. We want to ensure that the street signs we add do not overlap each other or overlap with the path itself. However, we cannot guarantee that this is always possible. In Table 6.7 we show how many conflicts between street signs occur on the paths we tested. This table also illustrates how many conflicts between street signs and road segments occur.

The total number of street signs which overlap each other is fairly small. In all experiments 192,262 street signs were placed. Only 3,963 of them were placed so that they overlap another street sign. This means only about 2.06% of all street signs overlap another street sign. Of all 192,262 street signs only 4,471 (2.33%) are drawn upon a road segment. These results indicate that the heuristic in place does perform very well. However, a refinement of this part may yield even better results.

Table 6.6: This table shows how many conflicts (i. e.,false intersection) were produced by the algorithm. Most of the paths tested produced no conflict. Only few produce more than 3 conflicts. Conflicts of the central path with the pre- and suffix path are not considered in this analysis.

Number of "false intersection"	0	1	2	3	4	> 4
Frequency	68.03 %	14.44 %	7.25 %	4.54 %	3.87 %	1.87%

Table 6.7: This figure reports how many conflicts c between a street signs occur and how many street signs have be drawn on top of a road.

#Conflicts c in one path	0	$(0, 5]$	[6, 10]	[11, 15]	> 15
Street Signs overlap each other	83.75 %	14.69 %	1.12 %	0.33 %	0.11%
Street Signs overlap a route segment	70.66 %	28.71 %	0.60 %	0.01 %	0.01 %

6.1.5 Running Time Analysis

As noted before, the schematization algorithm (c.f. Algorithm 2) is applied to the central path. In Figure 6.4 we illustrate a correlation between the number of vertices and the running time of the algorithm. Recall that the algorithms time complexity is $O(n^6)$ where n is the number of vertices. In the figure one black circle represents one run of the algorithm. The red curve shows a function of the for $f(n) = a + b \cdot n^6$. The coefficient b is very small. It is about $1.302 \cdot 10^{-10}$. Further, we can see that the function increases more sharply than the plotted experimental results indicate. This hints that a deeper, better analysis of the complexity of the schematization algorithm may yield a better upper bound. The gap between 45 and 80 vertices seems to be an coincidence. The figure shown is based upon 10,000 shortest path queries with source and target vertex selected uniformly at random.

In Figure 6.5 the results of the schematization algorithm with randomly generated monotone paths is depicted. We tested paths with lengths ranging from 10 vertices to 750 vertices, increasing the length of path by 10 vertices in each step. For every different path length multiple paths were generated.

The Figure 6.6 reports the total running time of our route sketch algorithm. The vast majority of the examples take less than 50ms on the computer which was used for evaluating.

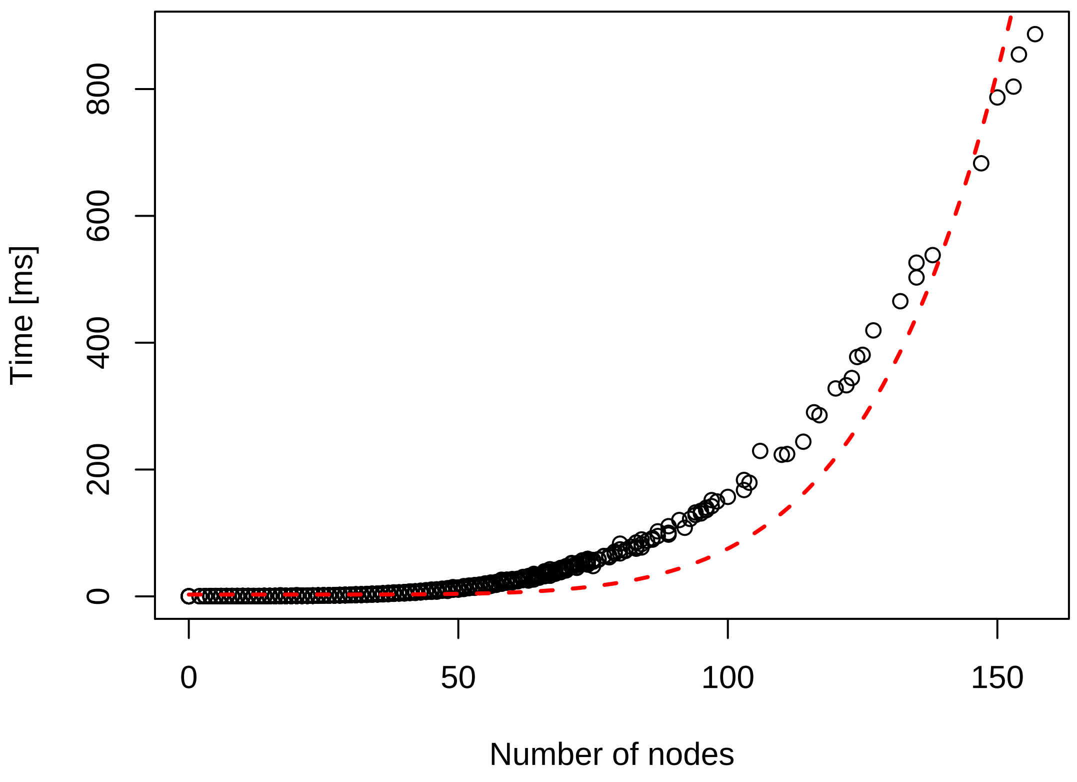


Figure 6.4: This figure reports the running time of the schematization algorithm in milliseconds according to the number of vertices on the Level-3 path. One black circle represents one run of the algorithm. The red curve shows a function which is in the form of $f(n) = a + b \cdot n^6$.

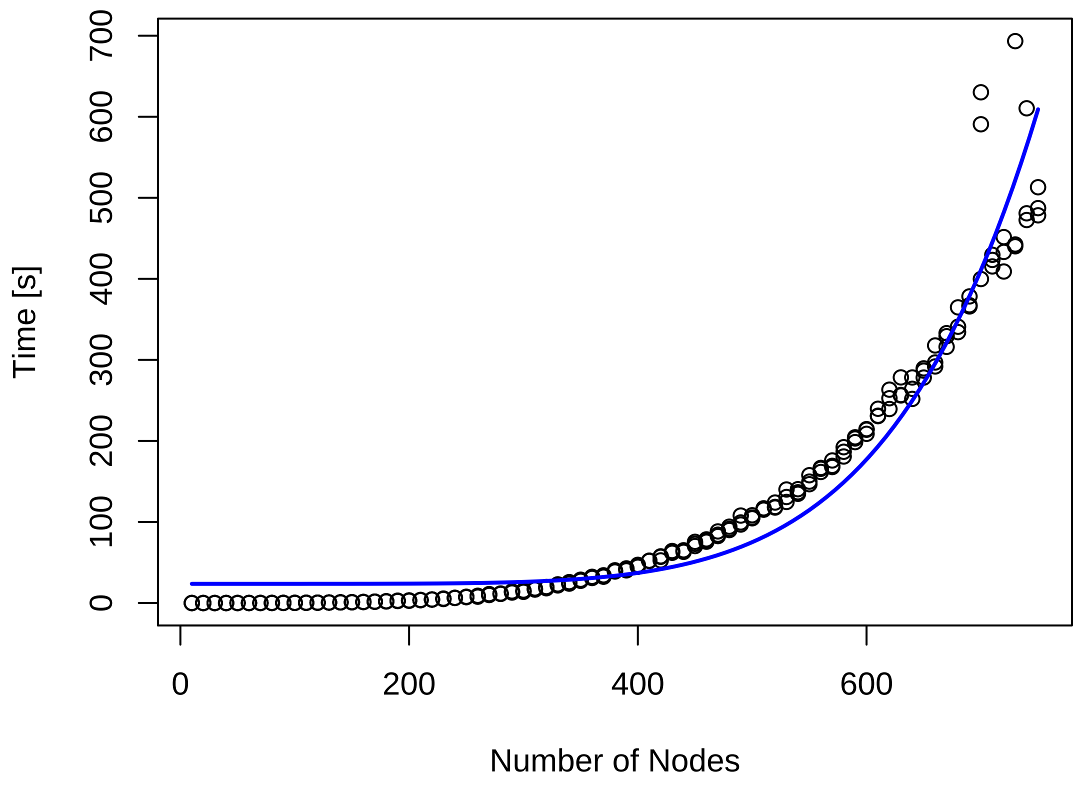


Figure 6.5: This figure shows how much time the schematization algorithm needs depending on the number of input vertices of the path. Tested were artificially generated monotone paths. Every black circle in the figure represents one run of the algorithm. The blue line is a function in of the form $f(n) = a + b \cdot n^6$.

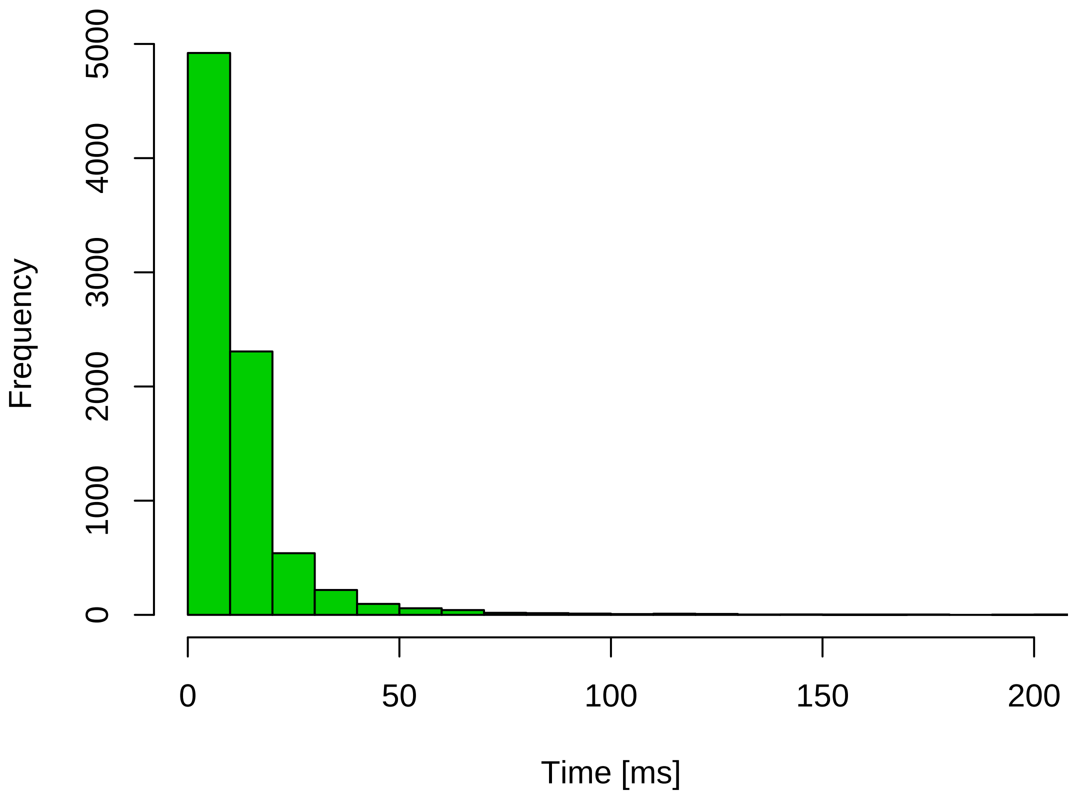


Figure 6.6: This figure shows the total running time of our implemented route map design algorithm. For the majority of the cases the total running time is below 50 ms.

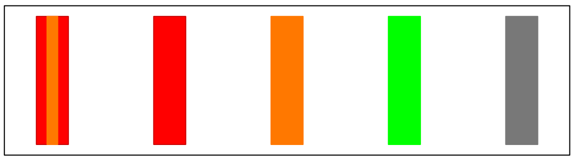


Figure 6.7: The colors for the categories are shown from left to right in decreasing order.

6.2 Case Studies

In the following we present some results regarding the visual representation of our route map sketch algorithm. Some parameters which can change the output of our route map sketch algorithm are changed and the created sketch is shown and discussed. The colors of the road identify its category. The higher the category is the more important the road is.

In the previous part of the experiments we discussed the possibility of changing the parameter ϵ for the Douglas-Peucker algorithm. Not only does that change the size of the input for the schematization algorithm but it also changes the appearance of the path. In Figure 6.8 we show a path which we chose to schematize. In Figure 6.10 we show the same path schematized with 10,000 as value for the parameter ϵ for the Douglas-Peucker algorithm. It is obvious that far less vertices have been removed. The highway "A4" has far more segments than depicted in Figure 6.9. A higher value for ϵ seems to be justified, because these additional road segments do not add value for the user.

Recall that we cannot remove all vertices of the path. There are certain vertices which the algorithm keep. These are either decision points or vertices that are necessary to avoid inconsistent turn directions. Thus, after the parameter has reached a certain value all vertices which can be removed are removed. All ϵ values above this threshold cannot change the resulting path. This can be observed by comparing Figure 6.9 where the value for ϵ is 100,000 with Figure 6.11 where the value for ϵ is 1,000,000. A difference in both figures is hardly noticeable.

The appearance of the schematization relies heavily on the set of allowed directions $\mathcal{C}$. Although Klippel [Kli03] could show that $\mathcal{C}_{45} = \{0°, 45°, 90°, \ldots, 315°, 360°\}$ of directions is sufficient for most people, we chose to use $\mathcal{C}_{30} = \{0°, 30°, 60°, \ldots, 330°, 360°\}$ as set of allowed directions for our approach.

In Figure 6.8 we show a path we chose to schematize. In Figure 6.9 we depict the schematization of the path with the set $\mathcal{C}_{30}$. In Figure 6.12 the same path is schematized, however, the set $\mathcal{C}_{45}$ of allowed directions is used. The difference that can be seen is noticeable. The lesser allowed directions the more skewed the schematized path is. We can see that the lesser allowed directions lead to a far more artificial appearing route. The use of $\mathcal{C}_{30}$ instead of $\mathcal{C}_{45}$ for our evaluation does only increase the route complexity a little but it is closer to the geographical shape of the route.

However, the minimum costs produced by our schematization algorithm is in both instances the same. This is due to the fact that we only deal with monotone paths and. This lets us group the edges into two different sets: h- and v-edges. The number of different vertical

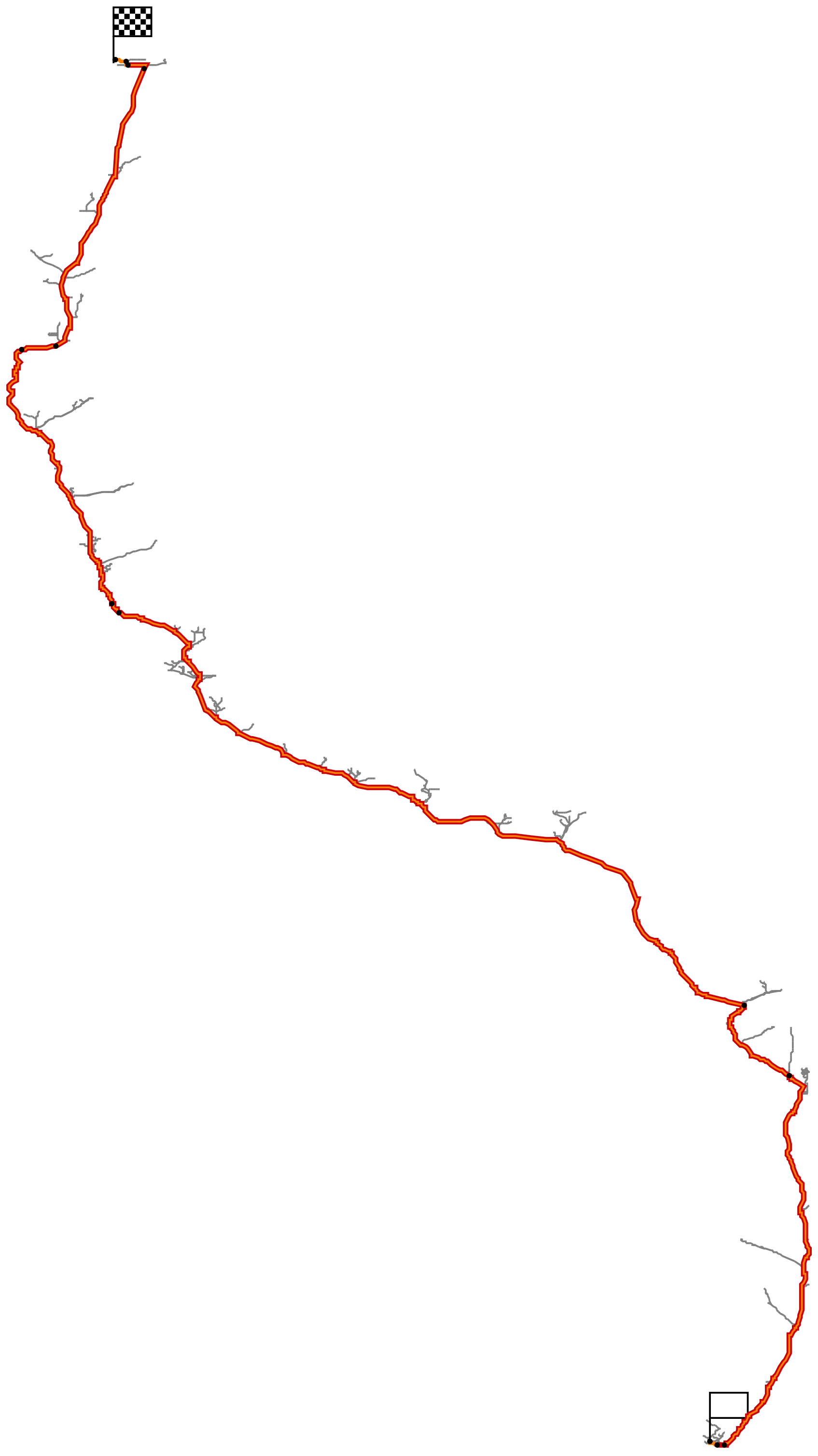


Figure 6.8: This figure depicts a route from the city of Karlsruhe to Gronau (Westf.). The distance of the route is about 450km. The route is highlighted with different colours corresponding to the street category. The streets depicted in grey are a caterpillar that surrounds the original route. Because everywhere the same scale is used to depict the route details at the beginning and at the end are hard to identify.

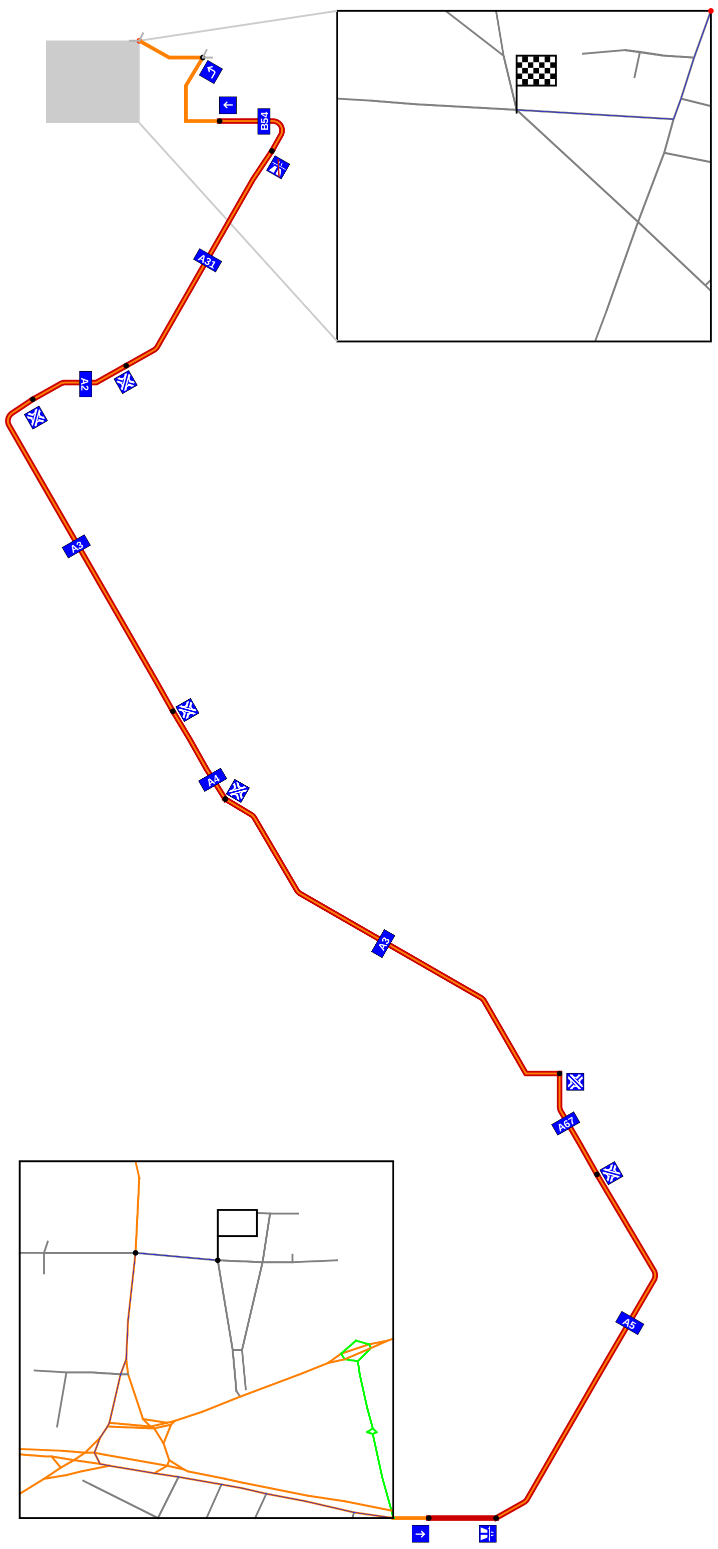


Figure 6.9: In this figure we show the same route as in Figure 6.8 but simplified with our route map design algorithm. The used set of allowed directions is $\mathcal{C} = \{0°, 30°, ..., 360°\}$, the parameter ϵ for the Douglas-Peucker algorithm is 100,000.

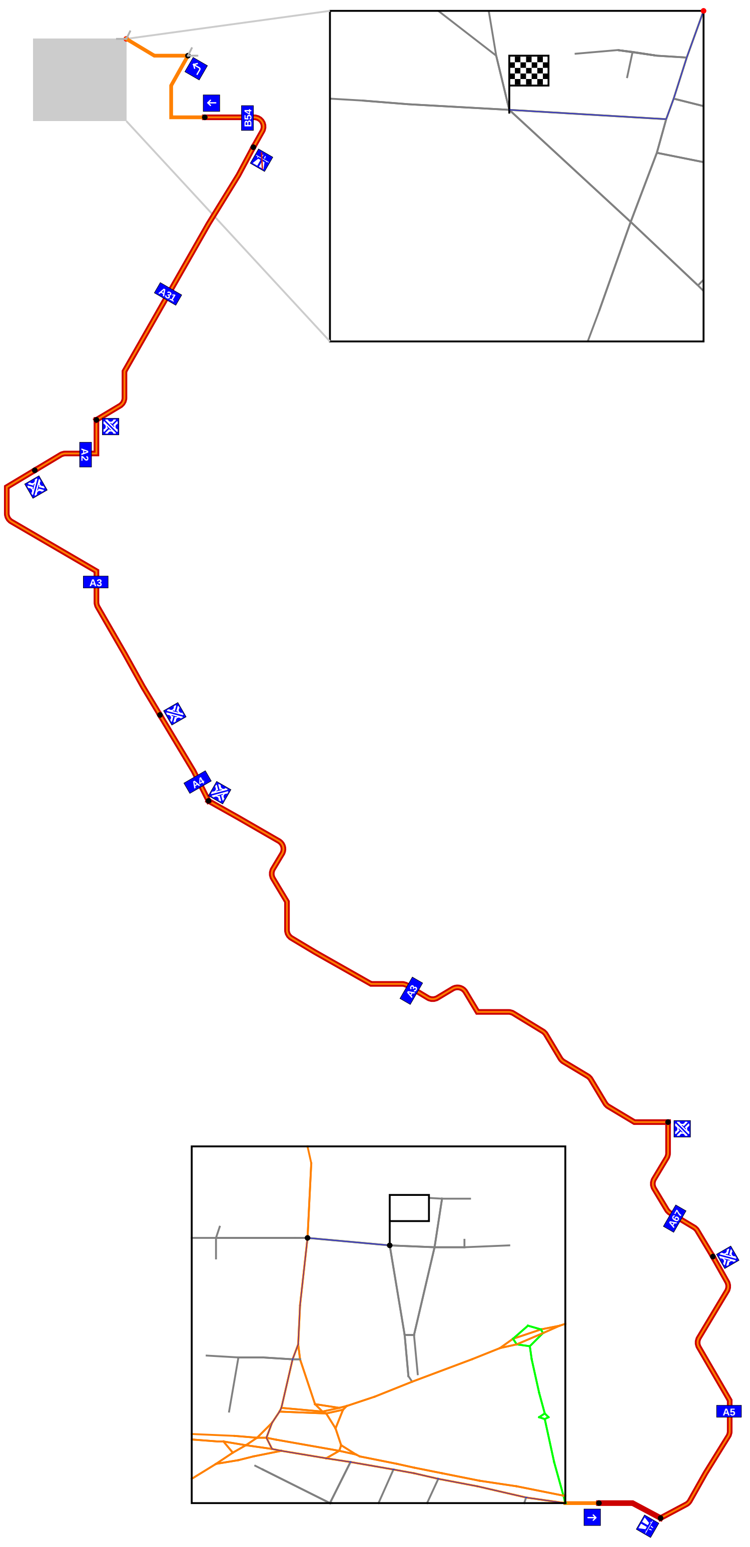


Figure 6.10: In this figure we show the same route as in Figure 6.8 but simplified with our route map design algorithm. The used set of allowed directions is $\mathcal{C} = \{0°, 30°, ..., 360°\}$, the parameter ϵ for the Douglas-Peucker algorithm is 15,000.

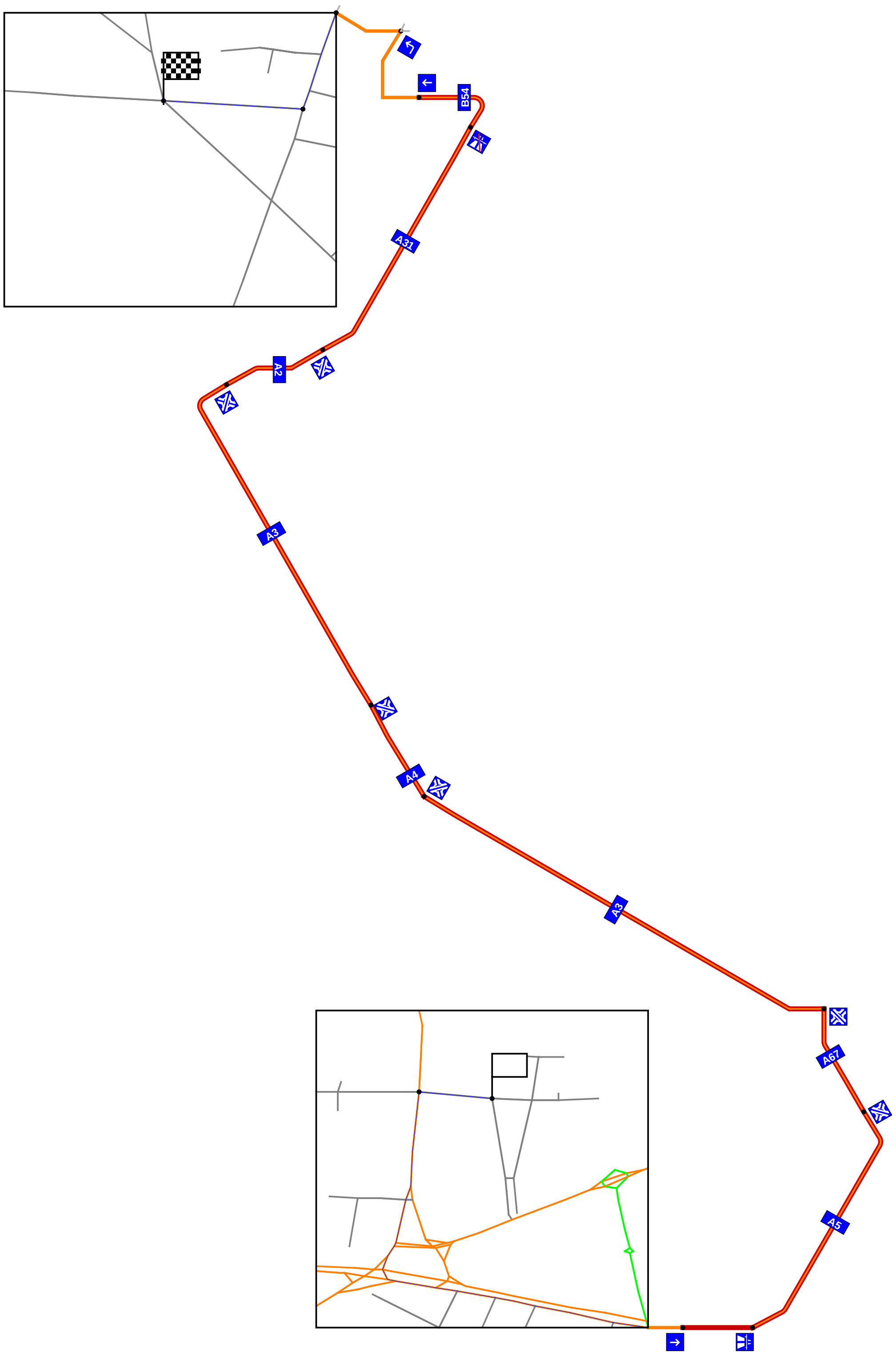


Figure 6.11: In this figure we show the same route as in Figure 6.8 but simplified with our route map design algorithm. The used set of allowed directions is $\mathcal{C} = \{0^\circ, 30^\circ, ..., 360^\circ\}$, the parameter ϵ for the Douglas-Peucker algorithm is 1,000,000.

directions in $\mathcal{C}$ does not influence the minimum cost. Thus, the results for the minimum cost do not differ.

Another parameter which influences the appearance of the path is the minimum edge length. We again schematize the path from Figure 6.8. In Figure 6.13 we schematized the same path with 100,000 as ϵ parameter for the Douglas-Peucker Algorithm. The set $\mathcal{C}_{30}$ is used as set of allowed directions. Instead of 50,000 we chose as minimum edge length the value 150,000. Because the size of the street signs depends on the minimum length, smaller minimum edge lengths lead to smaller street signs. The street signs can become very small and hard to read. Another parameter which we can alter is the minimum edge length. If it is small some edges stay very small. They may become very small and hardly noticeable. However, if the minimum edge length is very high very small edges appear very long. This may be irritating because path segments that are indeed very long and path segments that are very short might appear to be the same length. The depicted schematized paths illustrate this. Figure 6.13 shows an example where the minimum edge length is small compared to Figure 6.9 which has a larger minimum edge length. The higher minimum edge length is preferable. The streets leading to the destination are far better visible and the street signs are drawn larger.

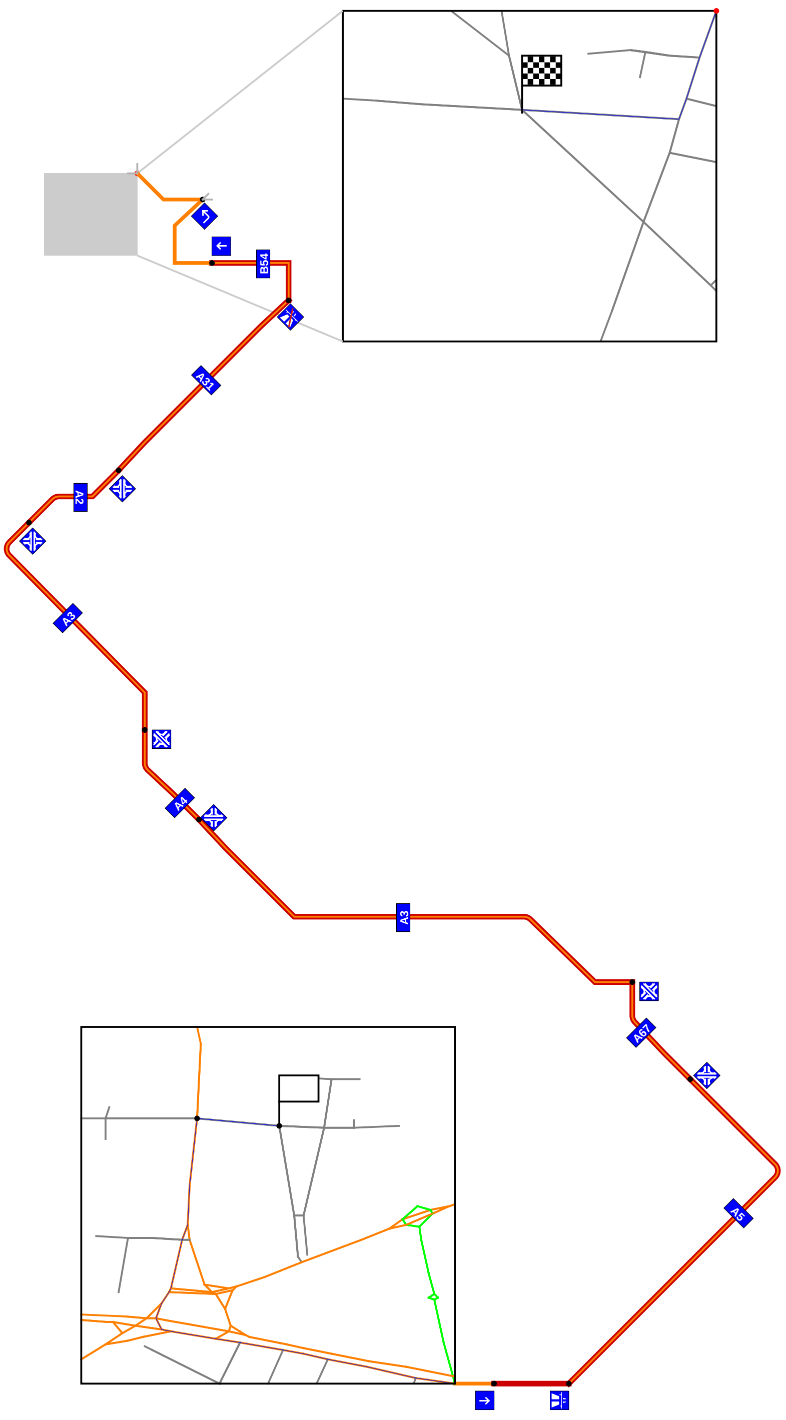


Figure 6.12: In this figure we show the same route as in Figure 6.8 but simplified with our route map design algorithm. The used set of allowed directions is $\mathcal{C} = \{0°, 45°, 90°, ..., 360°\}$, the parameter ϵ for the Douglas-Peucker algorithm is 100,000.

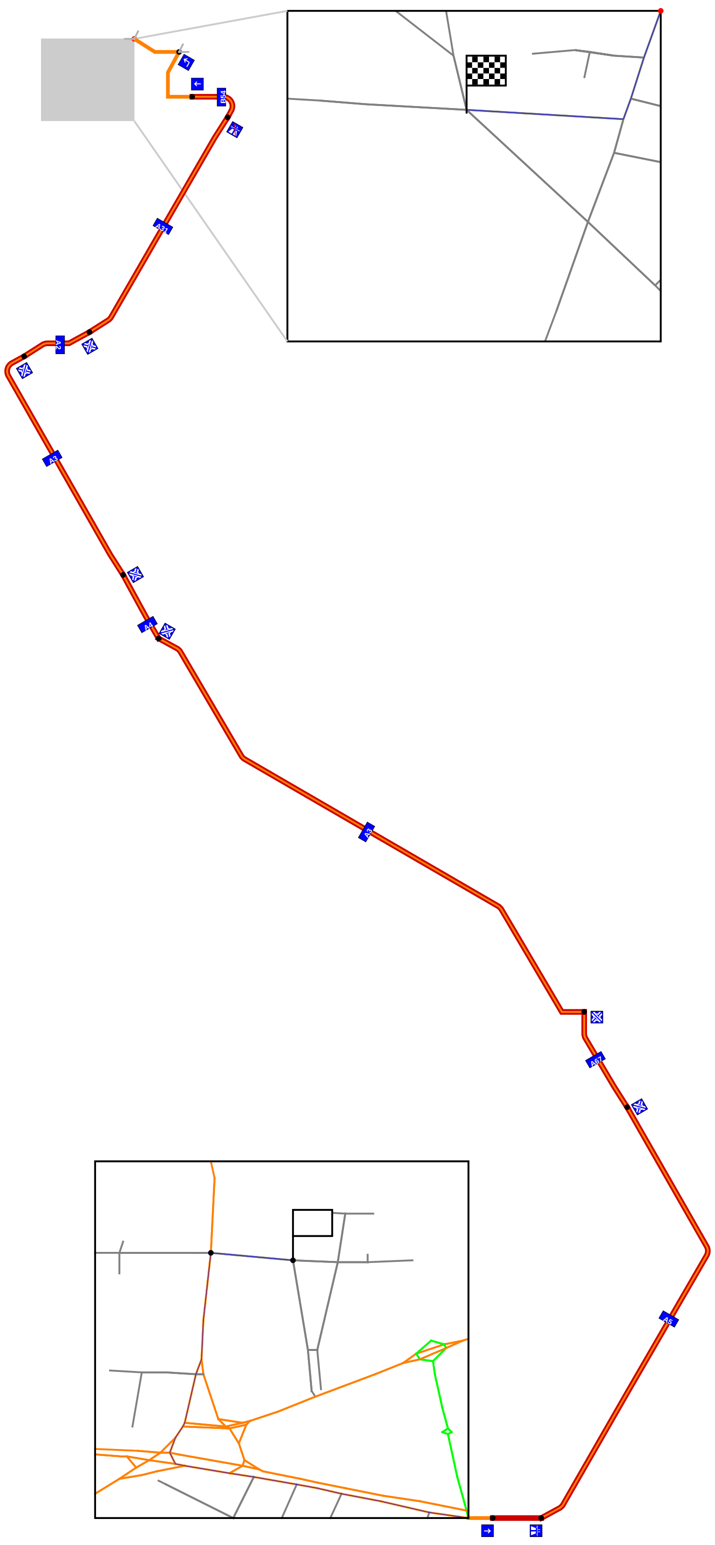


Figure 6.13: In this figure we show the same route as in Figure 6.8 but simplified with our route map design algorithm. The used set of allowed directions is $\mathcal{C}_{\ni\prime}$, the parameter ϵ for the Douglas-Peucker algorithm is 100,000. The minimum edge length is smaller than in Figure 6.8.

Chapter 7

Conclusion

In this thesis we could demonstrate how to schematize a given path in a road network. We motivated and introduced a schematization algorithm that is able to maintain the orthogonal order of an input path while minimizing the number of edges which cannot be assigned their preferred direction. This schematization algorithm restricts the edge direction of monotone paths to a given set of allowed directions and ensures that a minimum edge length is maintained. The analysis of the worst case running time that it depends only on the number of input vertices of the path and in no way on the number of allowed directions. We showed how to maintain the orthogonal order while schematizing a monotone path. Maintaining the orthogonal order was motivated by the way people have a mental picture of maps (c.f. mental maps).

Further, we were able to show how to use this schematization algorithm to schematize subsequent monotone paths of any given path. We illustrated how to resolve conflicts (i. e.,false intersections) of the schematized monotone paths with a very simple mechanism after they were attached to each other. This does no longer guarantee to preserve the orthogonal order of the full path, but our experiments showed that orthogonal order can can be maintained throughout the path with only very few excpetions. If violations occur they tend to affect only a small part of the node pairs. Finally, we showed how to place some street signs on the schematized path.

An evaluation of the devised algorithms is presented. We discussed several objective criteria for quality of the algorithm and presented a analysis of its time complexity.

7.1 Outlook

Multiple Paths. We could demonstrate how to schematize effectively a single path. In the future, it is desirable to devise an algorithm that is able to display multiple different schematized routes. The algorithms devised in this thesis can be a basis for this approach. However, all problems we dealt with in here can be more difficult to overcome when multiple paths are schematized. False intersections between path segments can happen, and avoiding them can become quite difficult. It is reasonable to assume that different paths from a given origin to a destination have some common edges. An algorithm which schematizes multiple paths has to be able to maintain true intersection. That is, intersections of the multiple paths. The solution presented to resolve conflicts of monotone paths will most likely need some modification.

Computing simple routes. The techniques in this thesis can be used to schmeatize any path. A different approach is to take one step backwards and to compute right from the beginning a route that has a low description complexity instead of being necessarily a shortest route. The hope is that such a route will be easier to follow and yield nicer schematizations.

If there is a path which is not considerable longer (perhaps only a certain percentage) but consists of only very little turns, this path may be much more favourable by a driver as he as much less decision points. Although the overall travel time might be a bit higher it is much more comfortable to drive such a route.

Decoration. The general route map mechanism we provided places several street signs as decorations. The method we used to place the street signs is a heuristic which proved, through experiments, to be fairly good. However, there are possibilities for improvement. The street signs could be placed via a sliding mechanism. So that there are no fixed positions for them, but they can slide along the edges of the path. Further, the size of the street signs should be maximized (but not overstep a certain upper limit) to enhance readability. But, on the same time the conflicts should not increase.

Another angle where improvements are possible is the placement of the pre- and suffix path (Level 1 and Level 5) if a conflict arises. A efficient data structure could be devised to answer queries for free space.

Although we enriched the information of our generated route maps, it may be useful to add further symbols or information to help the user. For example, it might be helpful to display distance information.

Positioning and displaying streetnames might also be helpful for finding the destination. However, efficiently placing streetnames is a difficult problem.

More efficient Schematization Algorithm. The schematization algorithm we provided has a time complexity of $O(n^6)$. Although we showed how to reduce the asymptotic running time to $O(n^5 \log n)$, this is still a considerable high running time. It is desirable to try and find another algorithm which is able to maintain the orthogonal order for monotone paths that has a lower asymptotic running time.

Appendix A

Schematization Algorithm

The following algorithms illustrate a more detailed version of the schematization algorithm.

Algorithm 7: calcOPT

```
1  for $i \leftarrow 1$ to $n - 1$ do
2      if $\{u_i, u_{i+1}\}$ h-edge then
3          $h(s_i) = 0$;
4      else if $\{u_i, u_{i+1}\}$ v-edge then
5          $h(s_i) = 1$;
6      else
7          $h(s_i) = \ominus$;
```

Algorithm 8: calcNmbV_Edges

```
   Data: Index i (upper bound), index j (lower bound)
1  vCounter = 0;
2  for $l \leftarrow i$ to $j$ do
3      index = $\mu^{-1}(l)$;
4      if $(y(u_i) \geq y(v_{index}) > y(v_{index-1}) \geq y(u_j))$ then
5          if $\{v_{index}, v_{index+1}\}$ v-edge then
6              vCounter++;
7      if $(y(u_i) \geq y(v_{index}) > y(v_{index+1}) \geq y(v_j))$ then
8          if $\{v_{index-1}, v_{index}\}$ v-edge then
9              vCounter++;
10 return vCounter;
```

Algorithm 9: cost_OneUpperStrip

```
Data: horizontal strip index $z$ with height set to 1
cost = 0;
for $l \leftarrow i$ to $z$ do
    index = $\mu^{-1}(l)$;
    if $\{v_{index-1}, v_{index}\}$ is $u_k$ crossing h-edge then
        cost++;
    if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing h-edge then
        cost++;
for $l \leftarrow z+1$ to $k-1$ do
    index = $\mu^{-1}(l)$;
    if $\{v_{index-1}, v_{index-1}\}$ is $u_k$ crossing v-edge then
        cost++;
    if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing v-edge then
        cost++;
return cost;
```

Algorithm 10: cost_OneLowerStrip

```
Data: horizontal strip index $z$ with height set to 1
cost = 0;
for $l \leftarrow k+1$ to $z$ do
    index = $\mu^{-1}(l)$;
    if $\{v_{index-1}, v_{index}\}$ is $u_k$ crossing v-edge then
        cost++;
    if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing v-edge then
        cost++;
for $l \leftarrow z+1$ to $j$ do
    index = $\mu^{-1}(l)$;
    if $\{v_{index-1}, v_{index}\}$ is $u_k$ crossing h-edge then
        cost++;
    if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing h-edge then
        cost++;
return cost;
```

Algorithm 11: cost_UpperLowerStrip

```
   Data: horizontal strip indeces $q, p$ with height set to 1
1  cost = 0;
2  for $l \leftarrow i$ to $q$ do
3      index = $\mu^{-1}(l)$;
4      if $\{v_{index-1}, v_{index}\}$ is $u_k$ crossing h-edge then
5          cost++;
6      if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing h-edge then
7          cost++;
8  for $l \leftarrow q+1$ to $k-1$ do
9      index = $\mu^{-1}(l)$;
10     if $\{v_{index-1}, v_{index}\}$ is $u_k$ crossing v-edge and $y(v_{index-1}) \geq y(u_p)$ then
11         cost++;
12     if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing v-edge and $y(v_{index+1}) \geq y(u_p)$ then
13         cost++;
14     if $\{v_{index-1}, v_{index}\}$ is $u_k$ crossing h-edge and $y(v_{index-1}) < y(u_p)$ then
15         cost++;
16     if $\{v_{index}, v_{index+1}\}$ is $u_k$ crossing h-edge and $y(v_{index+1}) < y(u_p$ then
17         cost++;
18 return cost;
```

Algorithm 12: Schematization Algorithm

```
1  $U$ = sort(P);
2  $n = |P|$;
3  for $i \leftarrow 1$ to $n-1$ do
4      Decide for all edges $(v_i, v_{i+1})$ if it is a h-edge or a v-edge;
5  calcOPT$(i, i+1)$;
6  for $i2 \leftarrow 2$ to $n-1$ do
7      for $i \leftarrow 0$ to $n - i2 - 1$ do
8          $j = i + i2$;
9          min_cost = $\infty$
10         for $k \leftarrow i+1$ to $j$ do
11             opt_cost = OPT$(i, k)$ + OPT$(k, j)$;
12             load horizontal strip heights $s[i, k]$ and $s[k, j]$;
13             join horizontal strip heights $s[i, j] = s[i, k] \cup s[k, j]$;
14             ul = k;
15             ll = k - 1;
16             for $l \leftarrow k-1$ to $i$ do
17                 if $s_l[i, k] == 1$ then
18                     $ul = l - 1$;
19                     break;
20             for $l \leftarrow k$ to $j$ do
21                 if $s_l[k, j] == 1$ then
22                     $ll = l - 1$;
23                     break;
24             for $q \leftarrow k$ to $ll$ do
25                 if $s_q[i, j] \neq -$ then continue;
26                 for $p \leftarrow k-1$ down to $ul$ do
27                     if $s_p[i.j] \neq -$ then continue;
28                     cost = cost_UpperLowerStrip(q, p) + opt_cost;
29                     if $min\_cost > cost_{l0}$ then
30                         min_cost = cost$_{l0}$;
31             for $l \leftarrow k-1$ down to $ul$ do
32                 cost$_u$ = cost_UpperStrip(l) + opt_cost;
33                 if $min\_cost > cost_u$ then
34                     min_cost = cost$_u$;
35             for $l \leftarrow k$ to $ll$ do
36                 cost$_l$ = cost_LowerStrip(l) + opt_cost;
37                 if $min\_cost > cost_l$ then
38                     min_cost = cost$_l$;
39             min_cost = calcNmbV_Edges(ul, ll);
40             min_cost = calcNmbV_Edges(i, j);
41         Save min_cost as $OPT(i, j)$ and the corresponding horizontal strip
           height assignment $s[i, j]$;
```

Appendix B

Examples

In the following we present some chosen examples to illustrate what our algorithm is capable of producing. For every path we schematized there is the original, unaltered path (i. e., the input path for our algorithm) with its original edge lengths and directions displayed.

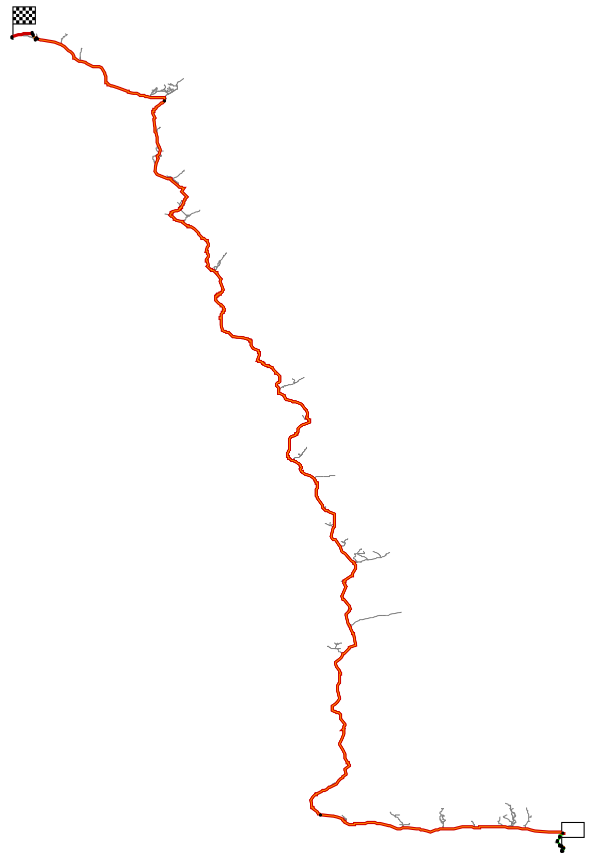


Figure B.1: Route of about 25km length.

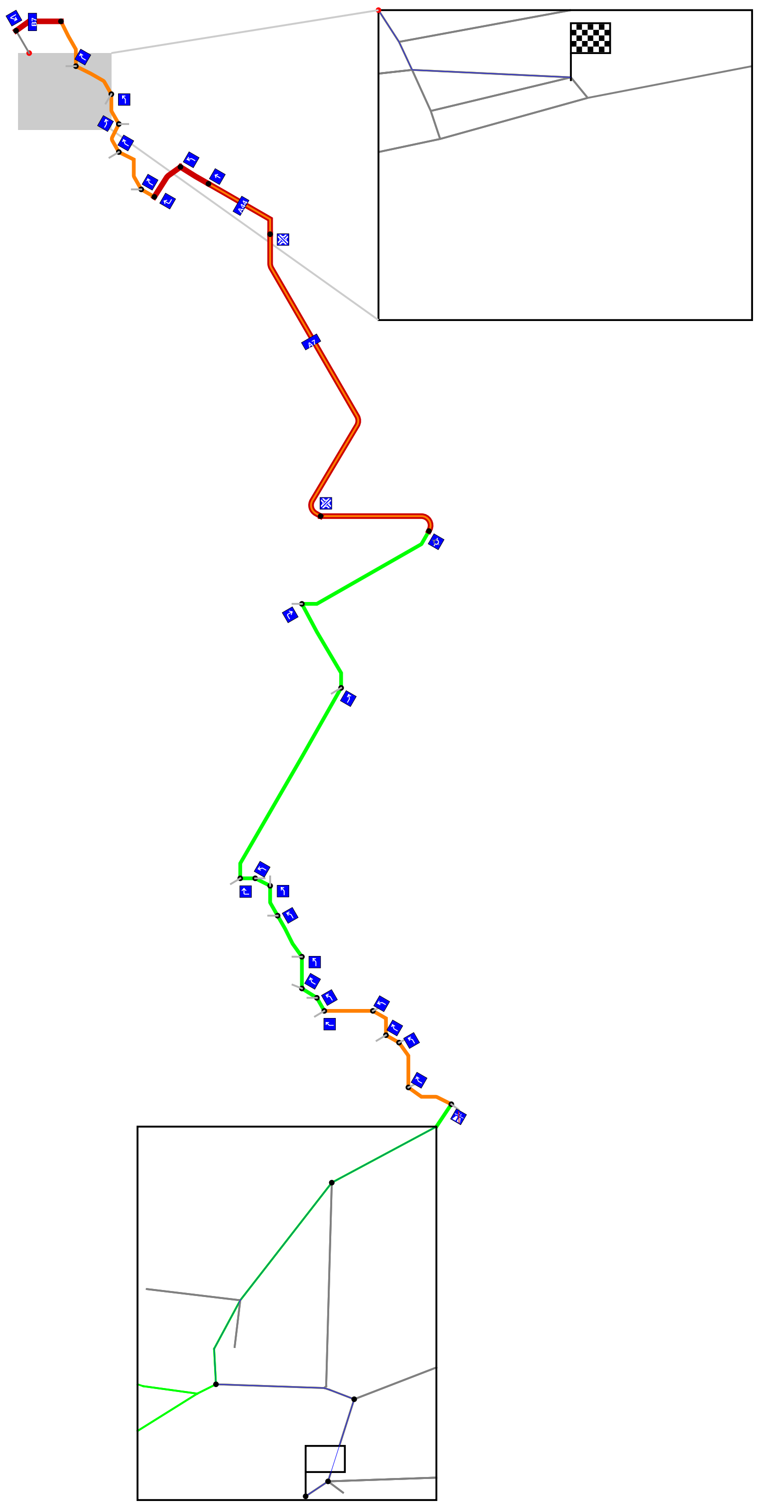


Figure B.2: In its schematized form the central path consists of 89 vertices.The value for the minimum length parameter is 150,000, the value for ϵ is 100,000.

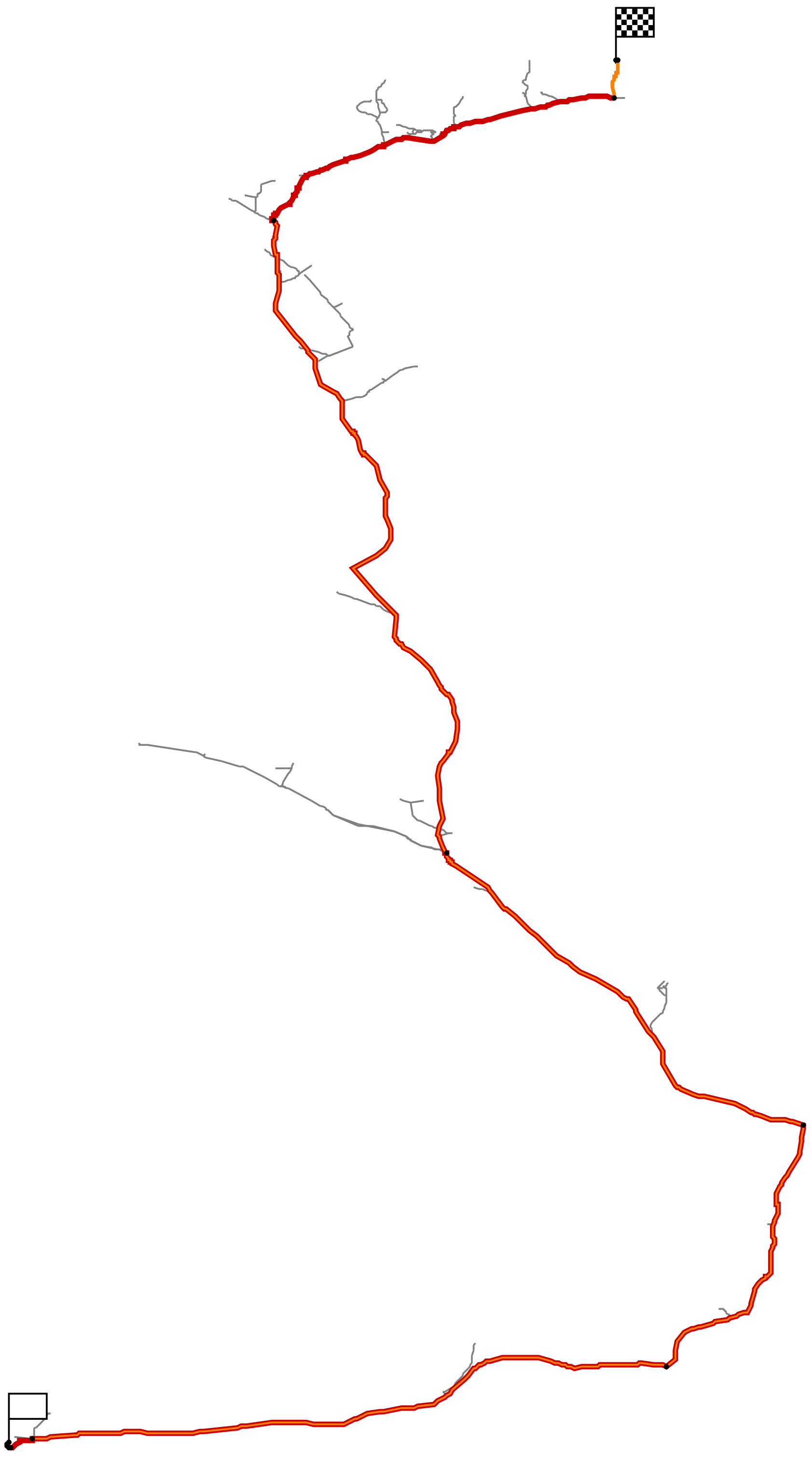


Figure B.3: Route of about 6km length.

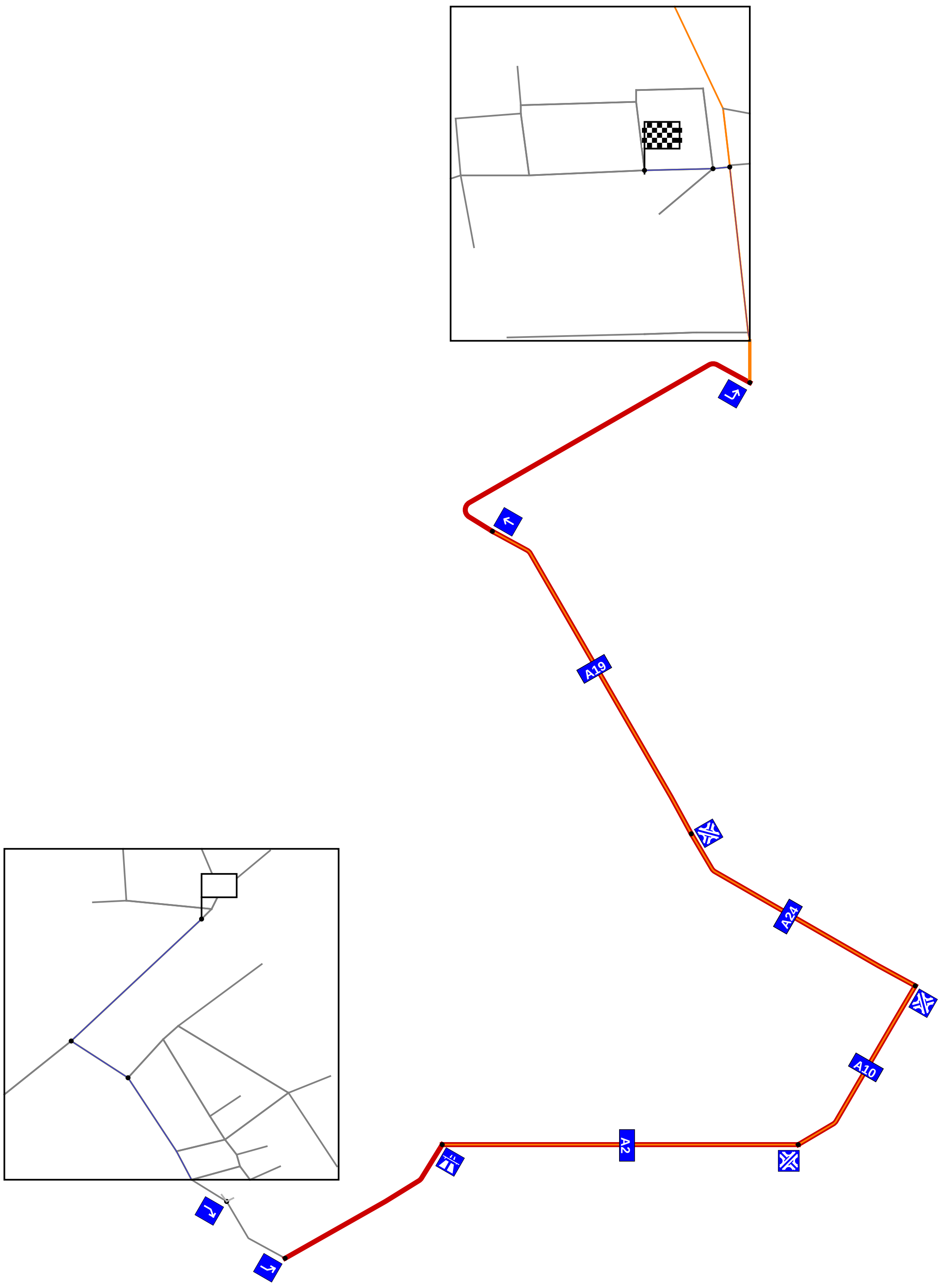


Figure B.4: In its schematized form the central path consists of 80 vertices.The value for the minimum length parameter is 150,000, the value for ϵ is 100,000.

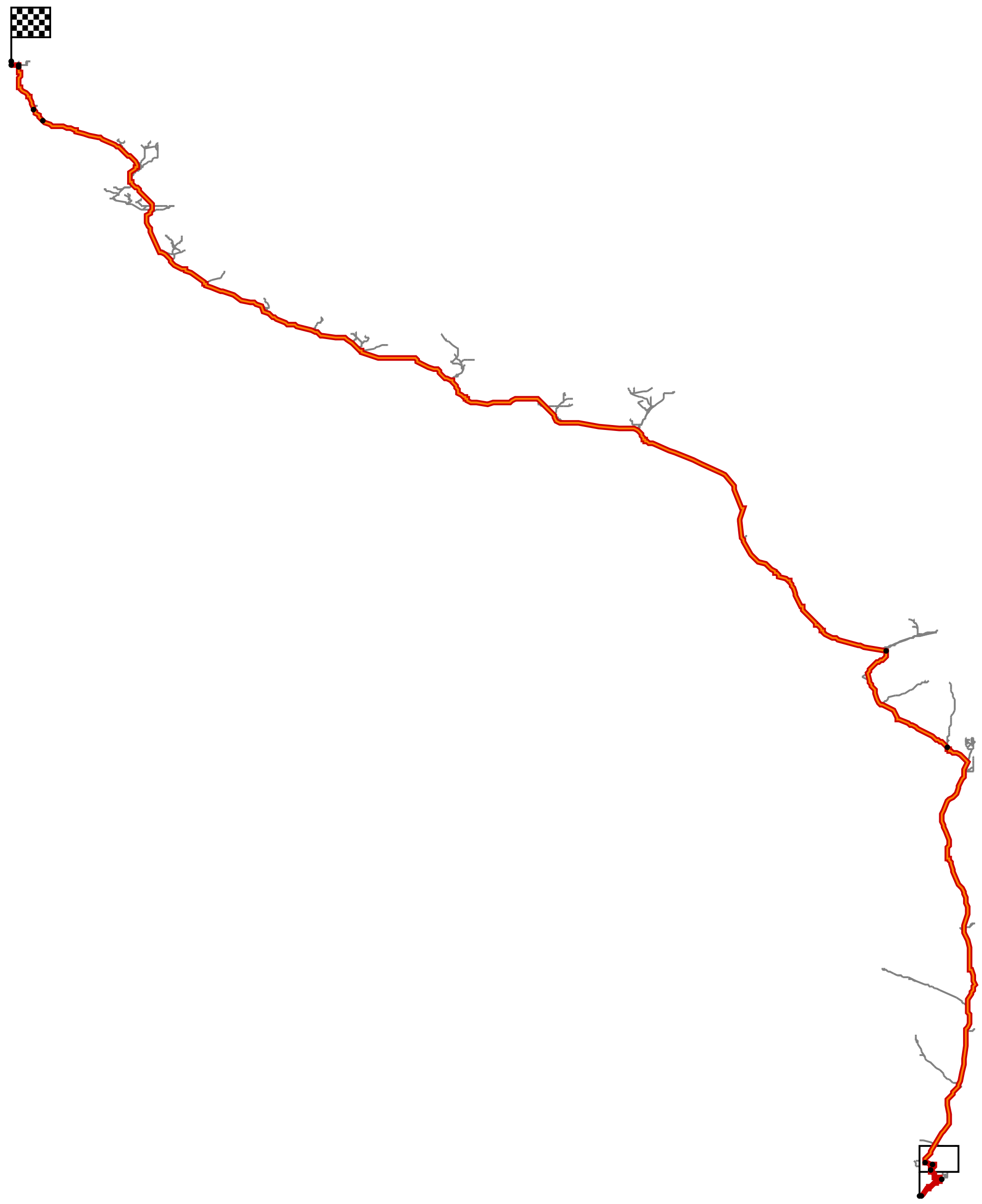


Figure B.5: Route of about 6km length.

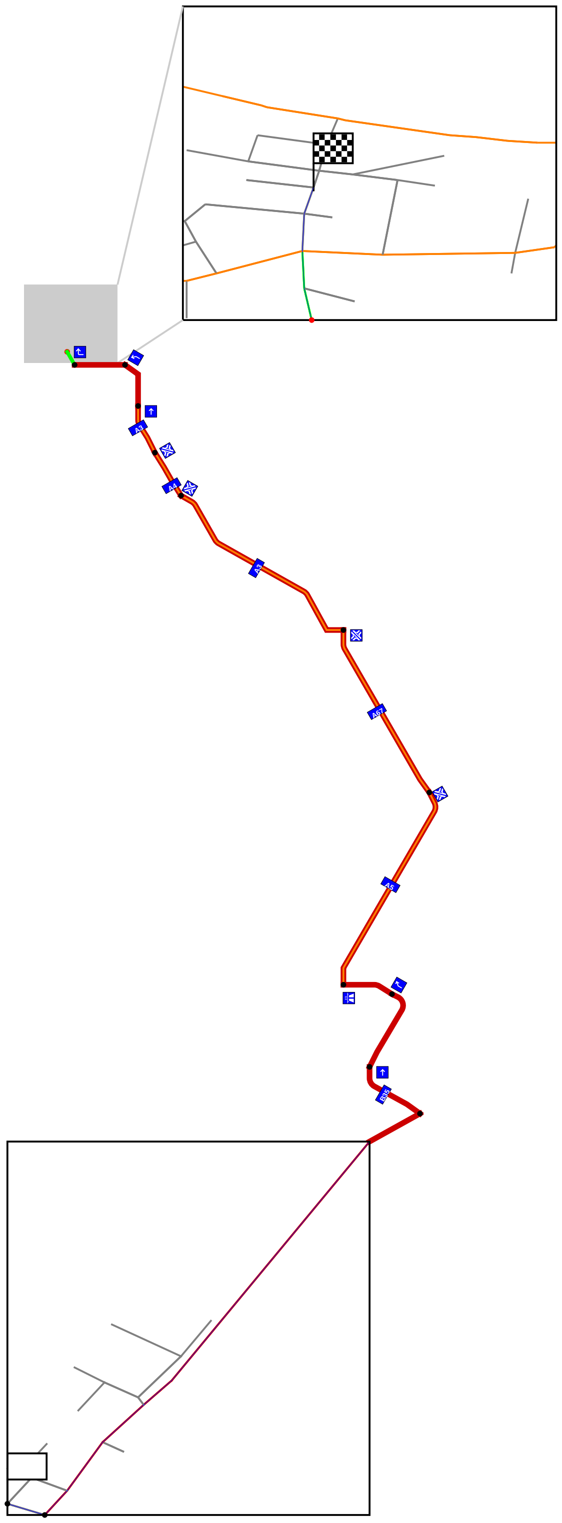


Figure B.6: In its schematized form the central path consists of 31 vertices. The value for the minimum length parameter is 150,000, the value for ϵ is 100,000.

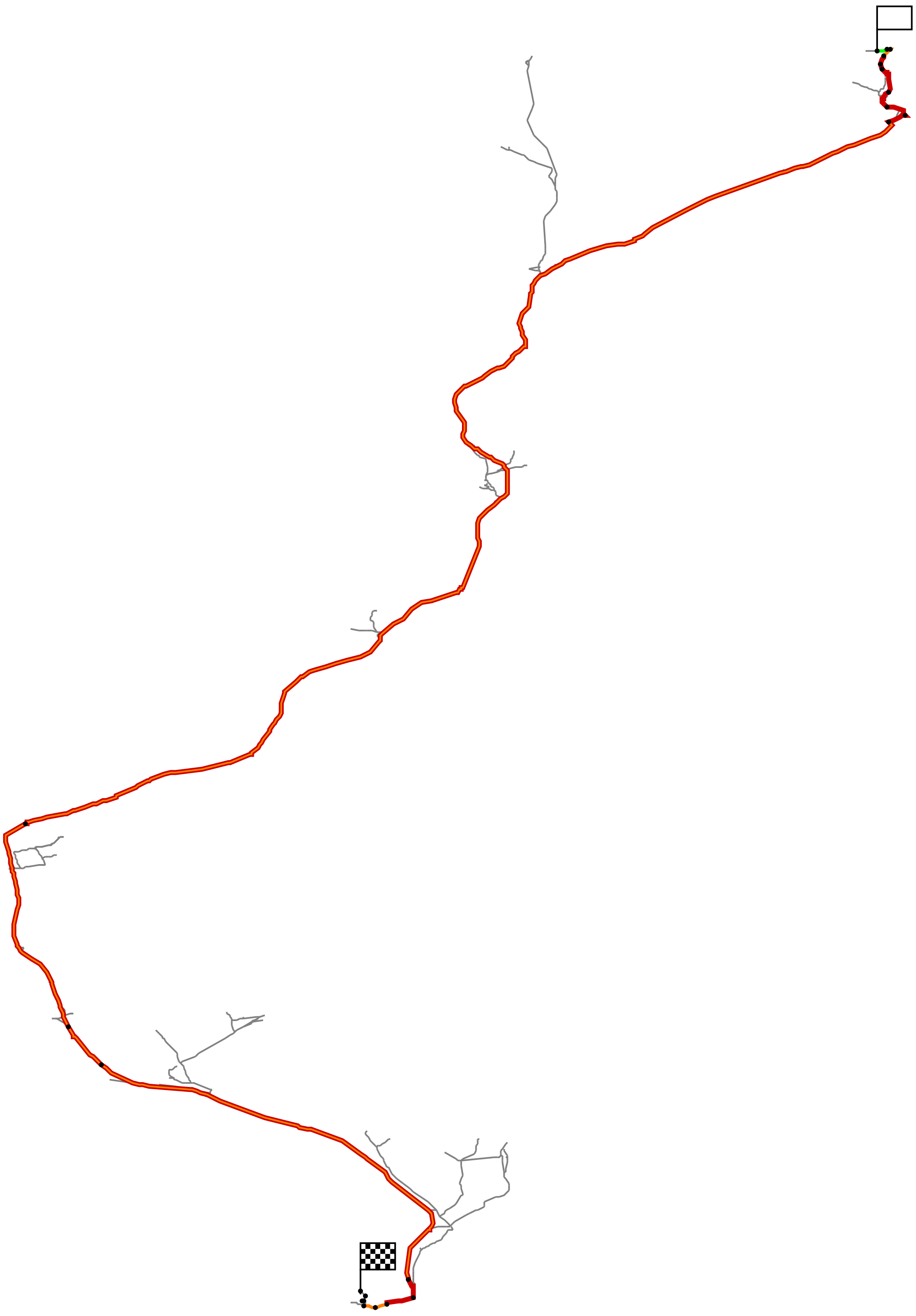


Figure B.7: Route of about 20km length.

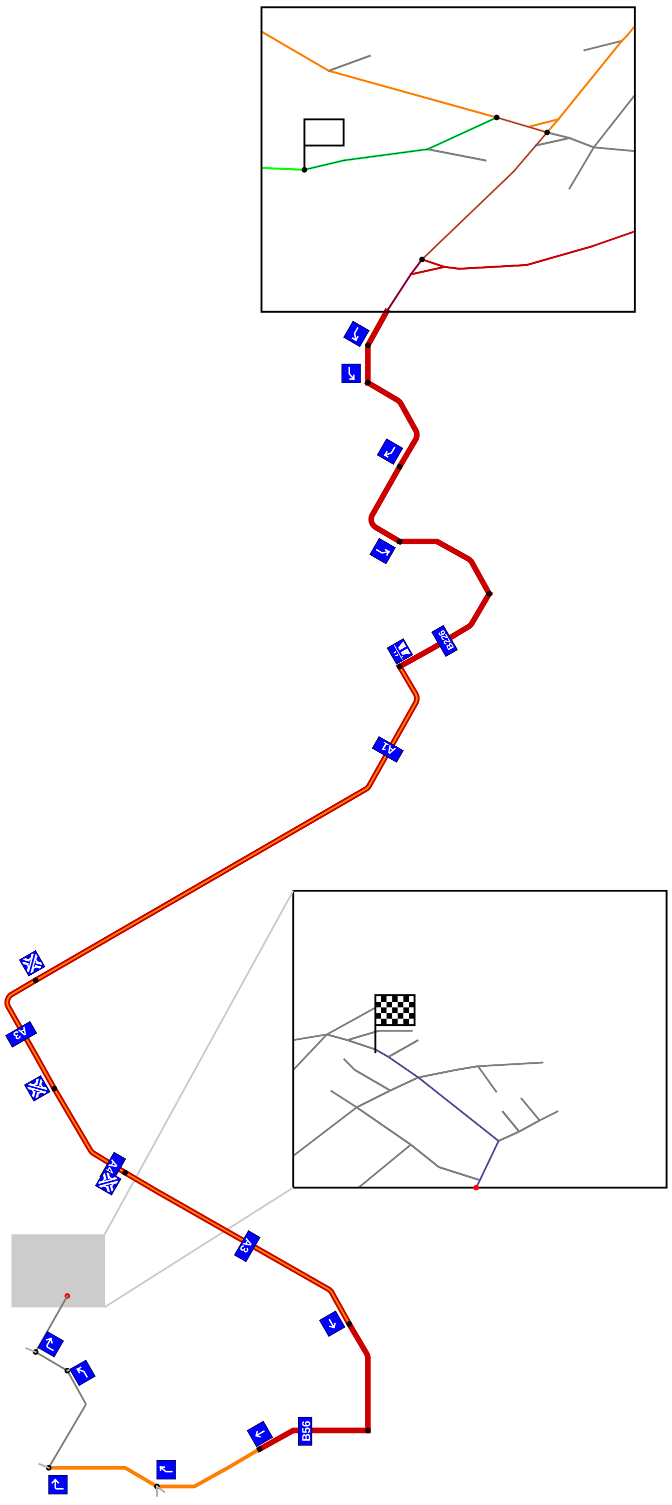


Figure B.8: In its schematized form the central path consists of 94 vertices. The value for the minimum length parameter is 150,000, the value for ϵ is 100,000.

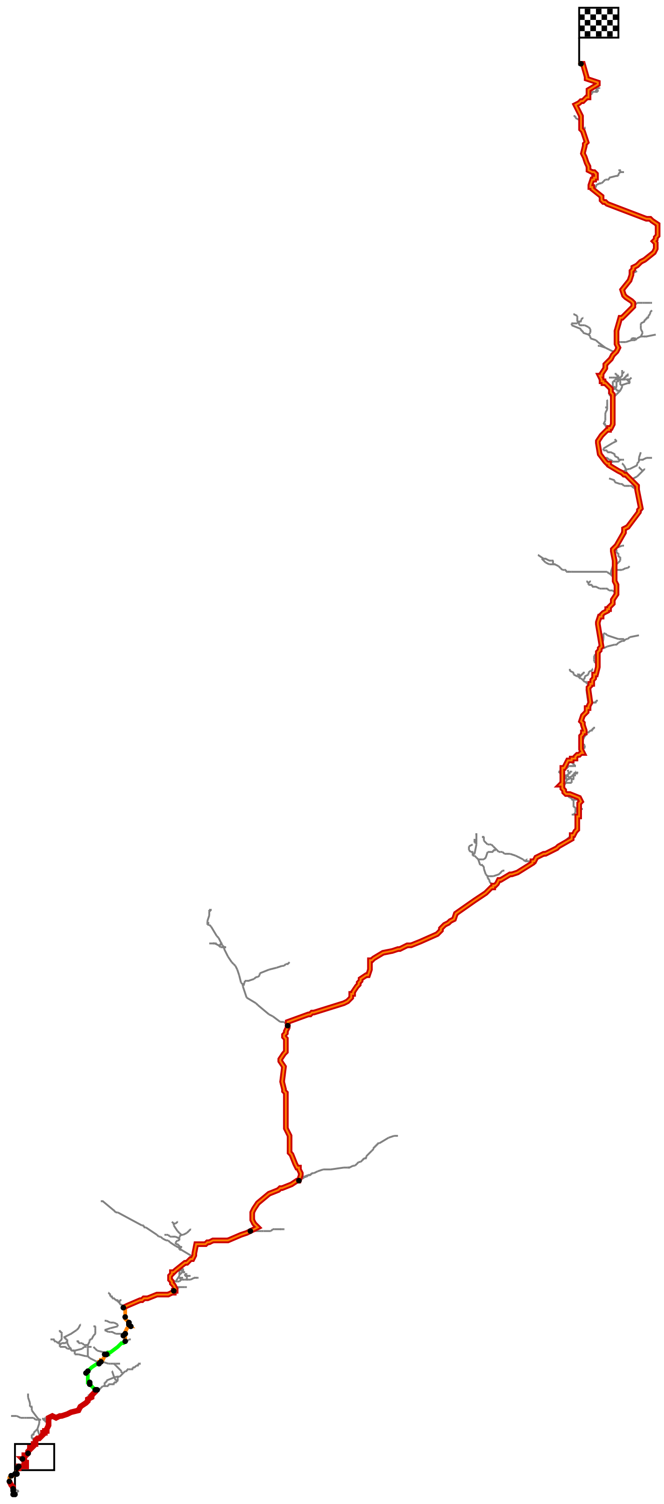


Figure B.9: Route of about 13km length.

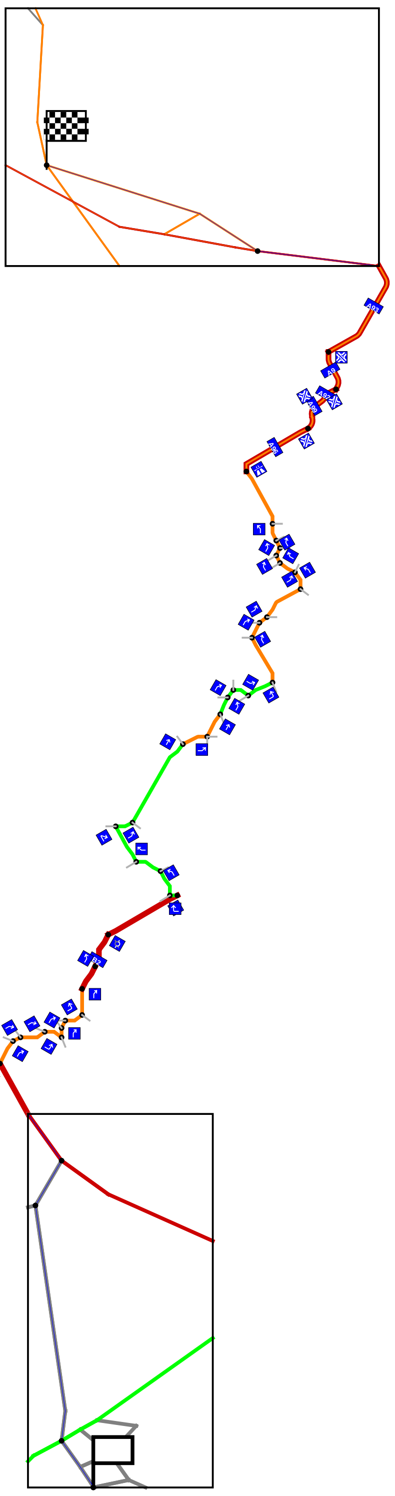


Figure B.10: In its schematized form the central path consists of 60 vertices. The value for the minimum length parameter is 150,000, the value for ϵ is 100,000.

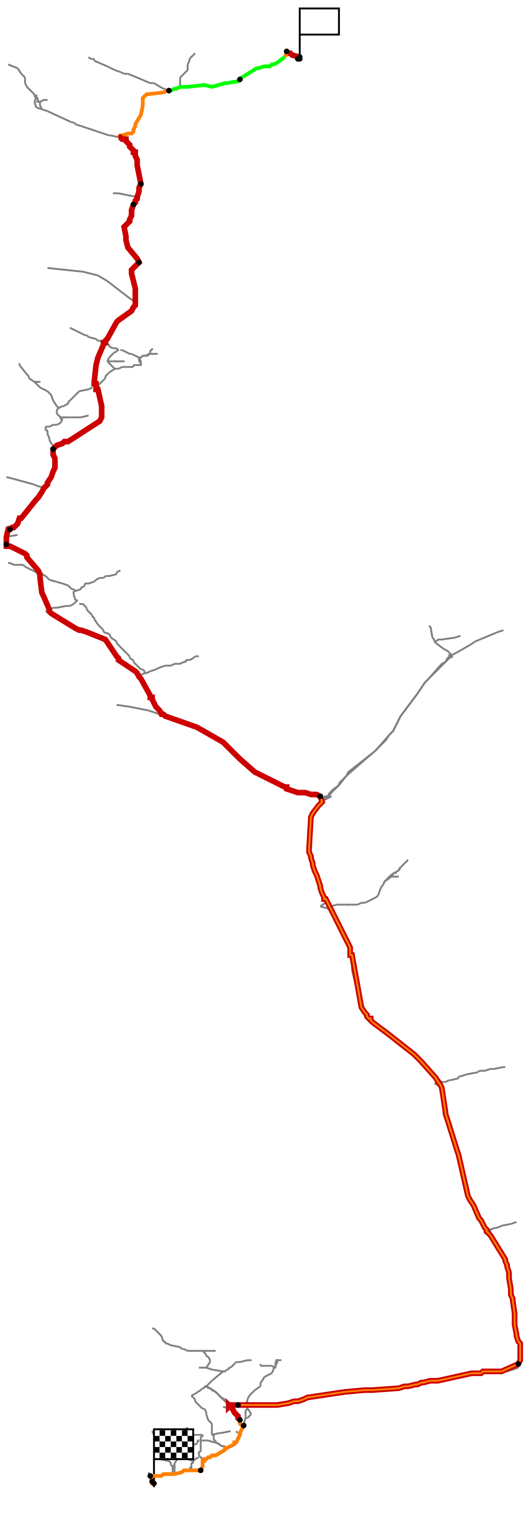


Figure B.11: Route of about 17,5km length.

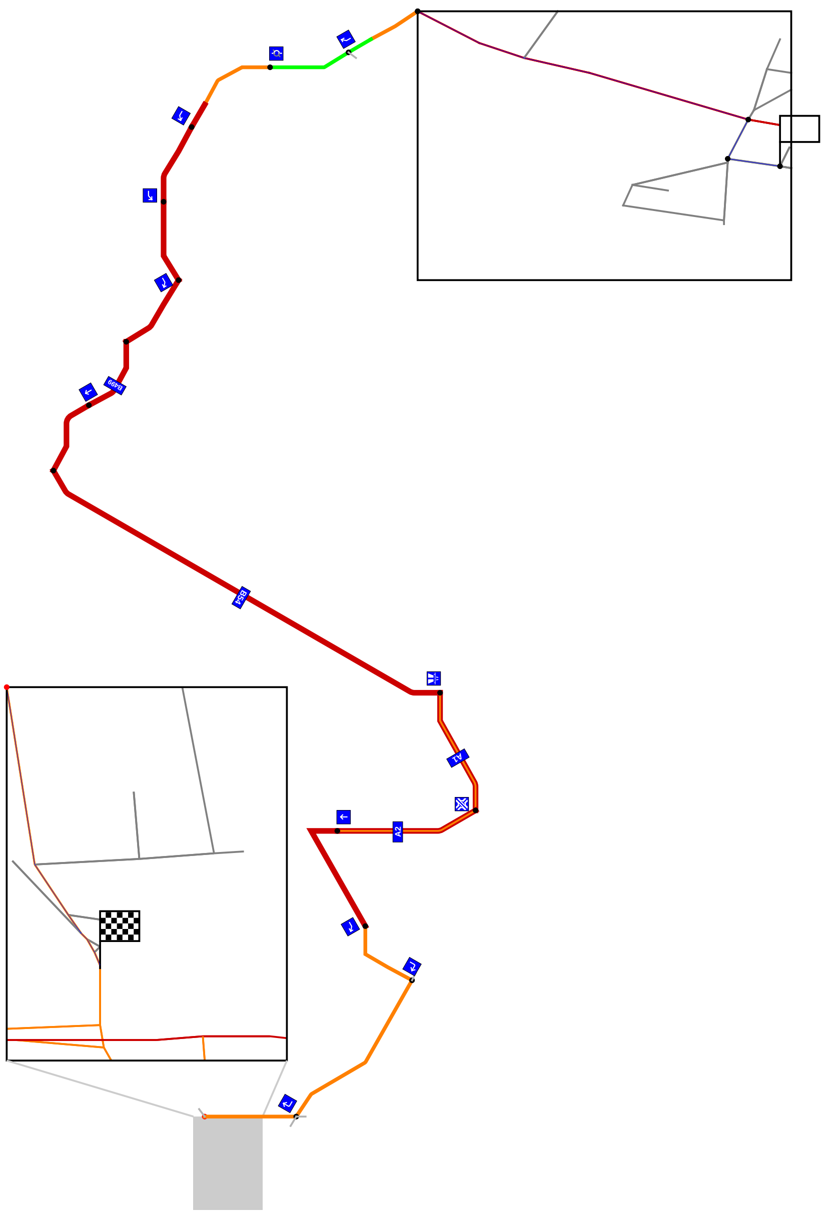


Figure B.12: In its schematized form the central path consists of 71 vertices. The value for the minimum length parameter is 150,000, the value for ϵ is 100,000.

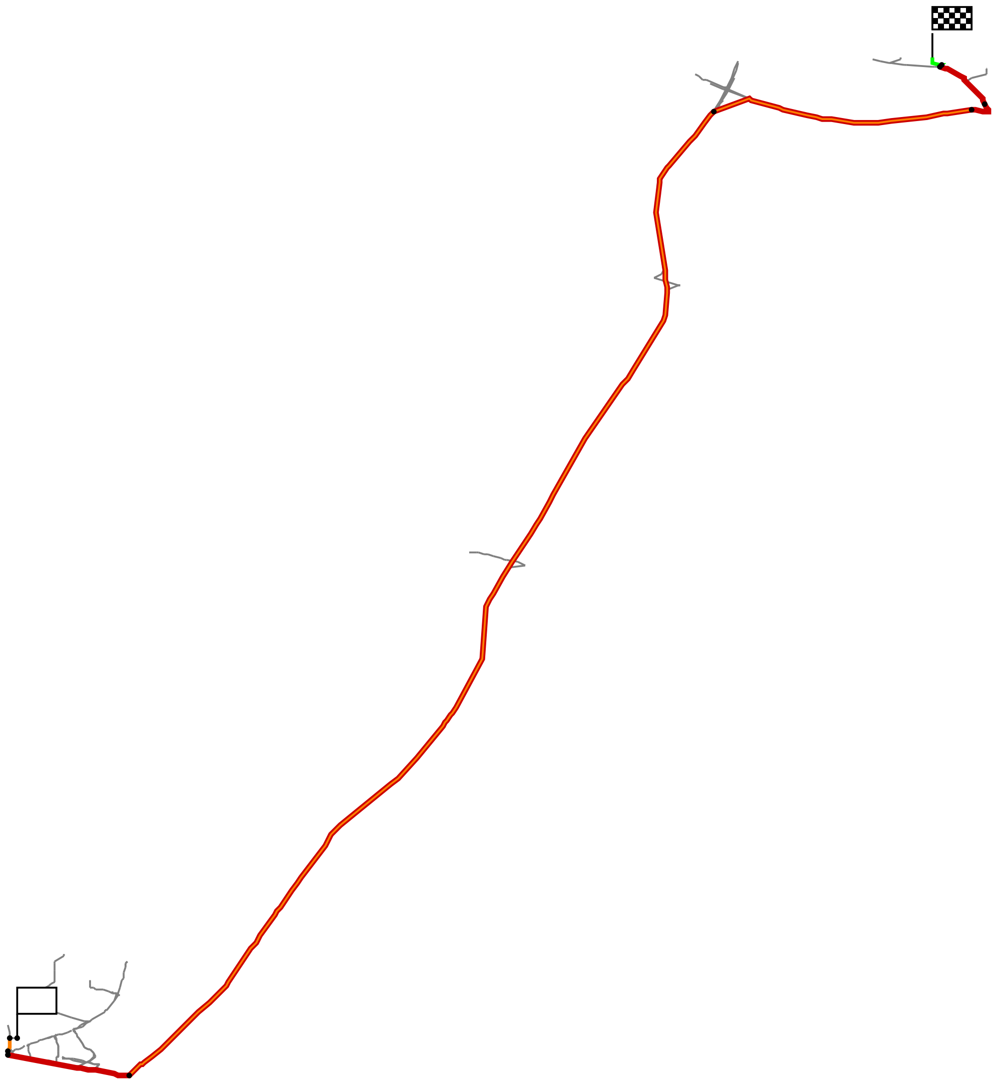


Figure B.13: In this figure a short route is shown from Karlsruhe to Wiesloch. Additional to the route some roads which cross the route segments are displayed. This is the same route which is displayed in Chapter 1.

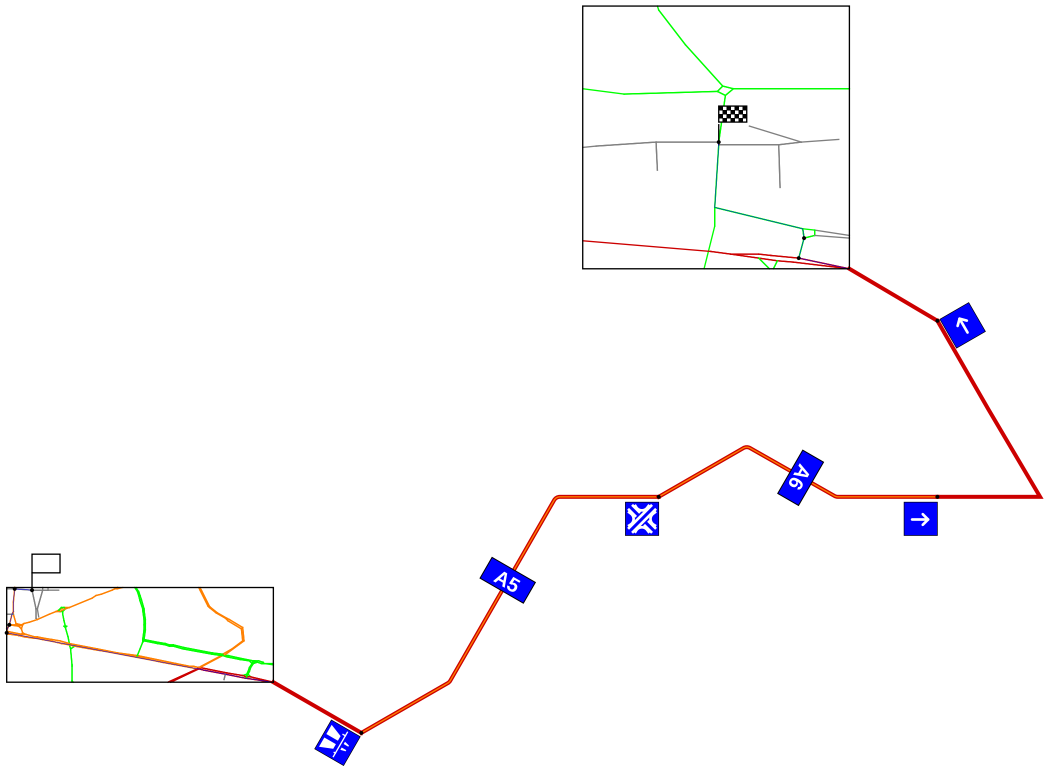


Figure B.14: This figure shows the schematized route from Karlsruhe to Wiesloch (see B.13) produced by our route map design algorithm.

Bibliography

[AF98] Mikhail J. Atallah and Susan Fox, editors. *Algorithms and Theory of Computation Handbook*. CRC Press, 1998. 47

[AG95] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995. 17

[AS01] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: improving usability through generalization. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 241–249, New York, NY, USA, 2001. ACM. 7, 18, 26

[ASSS86] M. D. Atkinson, J.-R. Sack, B. Santoro, and T. Strothotte. Min-max heaps and generalized priority queues. *Commun. ACM*, 29(10):996–1000, 1986. 61

[Bar02] Thomas Barkowsky. *Mental representation and processing of geographic knowledge: a computational approac*, volume 2541 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag New York, Inc., 2002. 13

[BD09] Reinhard Bauer and Daniel Delling. Sharc: Fast and robust unidirectional routing. *J. Exp. Algorithmics*, 14:2.4–2.29, 2009. 18

[Bel03] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003. 33

[BJ97] Marc Bernard and François Jacquenet. Free space modeling for placing rectangles without overlapping. *Jnl. Universal Comp. Sci*, 3:703–720, 1997. 62

[BLR00] Thomas Barkowsky, Longin Jan Latecki, and Kai-Florian Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In C. Freksa, W. Brauer, C. Habel, and K. F. Wender, editors, *Proc. Spatial Cognition II*, volume 1849 of *Lecture Notes on Computer Science*, pages 41–53. Springer Verlag, 2000. 13, 16

[BP09] Ulrik Brandes and Barbara Pampel. On the hardness of orthogonal-order preserving graph drawing. *Proc. of the 16th International Symposium on Graph Drawing*, 5417:266–277, 2009. 7, 17, 19, 28

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001. 33, 47, 50, 61 [dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer-Verlag; second edition, February 2000. 47

[DGHS93] David Dobkin, Leonidas Guibas, John Hershberger, and Jack Snoeyink, editors. *An efficient algorithm for finding the CSG representation of a simple polygon*, volume Volume 10, Number 1 / Juli 1993 of *Algorithmica*, New York, 1993. ACM. 25

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 11, 18, 65

[DL77] Stuart E. Dreyfus and Averill M. Law. *Art and Theory of Dynamic Programming*. Academic Press, Inc., 1977. 33

[DP73] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10:112–122, 1973. 13, 17, 23

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009. 11

[Far88] Gerald Farin. *Curves and surfaces for computer aided geometric design: a practical guide*. Academic Press Professional, Inc., San Diego, 1988. 13

[Fle95] Mark Fleischer. Simulated annealing: past, present, and future. In *WSC '95: Proceedings of the 27th conference on Winter simulation*, pages 155–161, Washington, DC, USA, 1995. IEEE Computer Society. 18

[FW91] Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *SCG '91: Proceedings of the seventh annual symposium on Computational geometry*, pages 281–288, New York, 1991. ACM. 60

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979. 17, 61

[GO97] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997. 45

[God91] M. Godau. Die Fréchet-Metric für Polygonzüge - Algorithmen zur Abstandsmessung und Approximation. Master's thesis, FU Berlin, 1991. 17

[HMdN06] Seok-Hee Hong, Damian Merrick, and Hugo A. D. do Nascimento. Automatic visualization of metro maps. *Journal of Visual Languages and Computing*, 17(3):203–224, 2006. 16

[HS92] John Hershberger and Jack Snoeyink. Speeding up the Douglas-Peucker Line-Simplification Algorithm. In *Proc. 5th Intl. Symp. on Spatial Data Handling*, pages 134–143, 1992. 25

[HSWW05] Martin Holzer, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Combining speed-up techniques for shortest-path computations. *J. Exp. Algorithmics*, 10:2.5, 2005. 18

[Kli03] Alexander Klippel. *Wayfinding Choremes - Conceptualizing Wayfinding and Route Direction Elements.* PhD thesis, University of Bremen, 2003. 12, 59, 75

[MELS95] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995. 13, 29

[MG07] Damian Merrick and Joachim Gudmundsson. Path simplification for metro map layout. In *Graph Drawing*, volume Vol. 4372, 2007. of *Lecture Notes on Computer Science*, pages 258–269, 2007. 16

[MHBB06] Tobias Meilinger, Christoph Hölscher, Simon J. Büchner, and Martin Brösamle. How much information do you need? Schematic maps in wayfinding and self localisation. In *Spatial Cognition*, pages 381–400, 2006. 12, 51

[Mil56] George Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956. 12

[MSS$^+$06] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra's algorithm. *J. Exp. Algorithmics*, 11:2.8, 2006. 18

[Ney99] Gabriele Neyer. Line simplification with restricted orientations. In *WADS '99: Proceedings of the 6th International Workshop on Algorithms and Data Structures*, volume 1663 of *Lecture Notes in Computer Science*, pages 13–24, London, UK, 1999. Springer-Verlag. 17

[NW06] Martin Nöllenburg and Alexander Wolff. A mixed-integer program for drawing high-quality metro maps. In P. Healy and N. S. Nikolov, editors, *Proc. 13th International Symposium on Graph Drawing (GD'05)*, volume 3843 of *Lecture Notes in Computer Science*, pages 321–333. Springer-Verlag, 2006. 16

[Ram72] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, November 1972. 13, 17

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, June 1998. 47

[SR04] Jonathan M. Stott and Peter Rodgers. Metro map layout using multicriteria optimization. In *Proc. 8th Internat. Conf. Information Visualisation (IV'04)*, pages 355–362. IEEE, 2004. 16

[SVA00] Tycho Strijk, Bram Verweij, and Karen Aardal. Algorithms for maximum independent set applied to map labelling. Technical Report UU-CS-2000-22, Department of Information and Computing Sciences, Utrecht University, 2000. 60, 61

[Tve92] B. Tversky. Distortions in cognitive maps. *Geoforum*, 23(2):131–138, 1992. 13

[Yam07] Lorena Yamamoto, Camara. Fast point-feature label placement algorithm for real time screen maps. In *Information Visualization*, volume 6, pages 249–260, 2007. 60