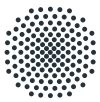

Universität Stuttgart

Labeling Interactive Maps

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Filip Krumpe

aus Berlin-Neukölln

Hauptberichter: Prof. Dr. Stefan Funke

Mitberichter: Prof. Dr. Dirk Burghardt

Tag der mündlichen Prüfung: 22.7.2020

Institut für Formale Methoden der Informatik

2020

Contents

Part I.

Prologue

1. Preface

Digital, geographic maps play a central role in the modern everyday life. When starting a day, one checks the maps app for possible traffic incidents on the way to the workplace (at least if one is going there by car). At lunch time, one is using digital maps to explore the surrounding to find a place to eat. Having finished work, one may aim for finding a suitable shop for doing the procurements and at the same time checking the status of the traffic on the roads. In the evening, one may be looking for a trendy bar in town to meet one's friends what requires to check the map again. For the weekend one has to find a place to travel to and relax, and something around to see and tell about one's colleagues on Monday. For all of this your maps provider is your best friend, making online map services one of the most important services provided by smartphones and tablets today.

In former times, good maps were the key technology to rule the world. Even a few decades ago, car navigation systems and updates were sold to customers at high prices. Today quite good data sets of geographic information are available for free. Map services like google maps changed the way we use and think about the value of geographic information. Nowadays it is something which is constantly available, for free and of very high precision.

Cartography and the creation of paper maps and globes were an art form not that long ago. The main challenge was to put as much information onto the map as possible, thereby maintaining clear association of the label to its corresponding feature and a good readability of the map. In practice, two interrelated problems occur. One had to choose from the huge set of data and find a proper subset of maximum size which can be displayed. On the one hand, the selection limits the space available for the label placement. On the other hand, the space available for label placement restricts the selection. If no space is available for placing the label, the corresponding element can not be displayed. Hence creating a paper map in former times always meant having to restrict the available data.

Today's digital technologies are allowing an elegant solution to this problem. By enabling the user to interact with the map view, i.e. panning and zooming the map view, these problems can partially be bypassed. If a user wants to see more details of a region, he can zoom in to get more details. Panning can be used to move the current viewport. For getting an overview of the surrounding of the current view, zooming out is the operation of choice. The corresponding gestures

internalized: Use a finger slide to pan or shift the map view or drag it if using a mouse. To zoom in use a two finger pinch or the mouse wheel.

In addition to this exploratory use, digital maps are also used to navigate while driving with the car or cycling or even walking. In this setting, it is common to align the map with the direction. So directions as seen on the map are corresponding to reality, i.e. turning right on the screen corresponds to turning right on the road. So this particular setting requires an additional mode of interaction: rotation. Furthermore it is usual that the map is always properly zoomed. Which means that while driving through a city, a very zoomed in view is presented. On the other hand, when driving in the back country a much more zoomed out view is appropriate. So the presented map always shows the significant part of the route ahead.

A map always shows a perspective to the world which we are not used to: a bird's eye view. To bring together the map view and the reality we need to identify parts of the map and link them to real entities. Anchors of orientation may be the visualized shapes of large area entities or the course and intersection of important roads and others. Examples may be the national borders, forests, lakes, the shape of rivers or the crossing of highways on a large scale map. This identification is possible for area and line features. It is difficult for a third kind of geographic features: point-like features. This may be a historic site or a human settlement on a small-scaled map. An identification of these can be done by labeling the features with their names (e.g. in case of cities) or icons (e.g. cafes, ATMs, …). In order to ensure a clear assignment and good readability, the displayed labels must not intersect and need to be aligned to the features they are naming.

Considering map interaction and labeling combined induces some additional problems. Panning a given map view is easy: the labels may slide in or out of the map view. When it comes to the other interactions things get much more difficult. What if the map view is zoomed? Obviously, when zooming out, things get smaller. So distances between the labeled entities shrink. In order to keep the labels readable, these are usually kept at constant size. But this induces a problem since labels might start overlapping each other. The two solutions available are: We either shrink the font size of the labels - but this will make them unreadable sooner or later. If we are keeping the font sizes constant, we need to remove some labels in order to restore readability. In the second case, it is intuitive to remove less important labels first so that more important labels of bigger cities for example can be kept visible for as long as possible.

Let's consider a labeling of human settlements for an example. A naive solution for reducing the amount of labels when zooming out would be to successively filter out labels of lower importance. The unintended effect would be that labels in sparsely populated regions would vanish early. Thus, creating unlabeled areas

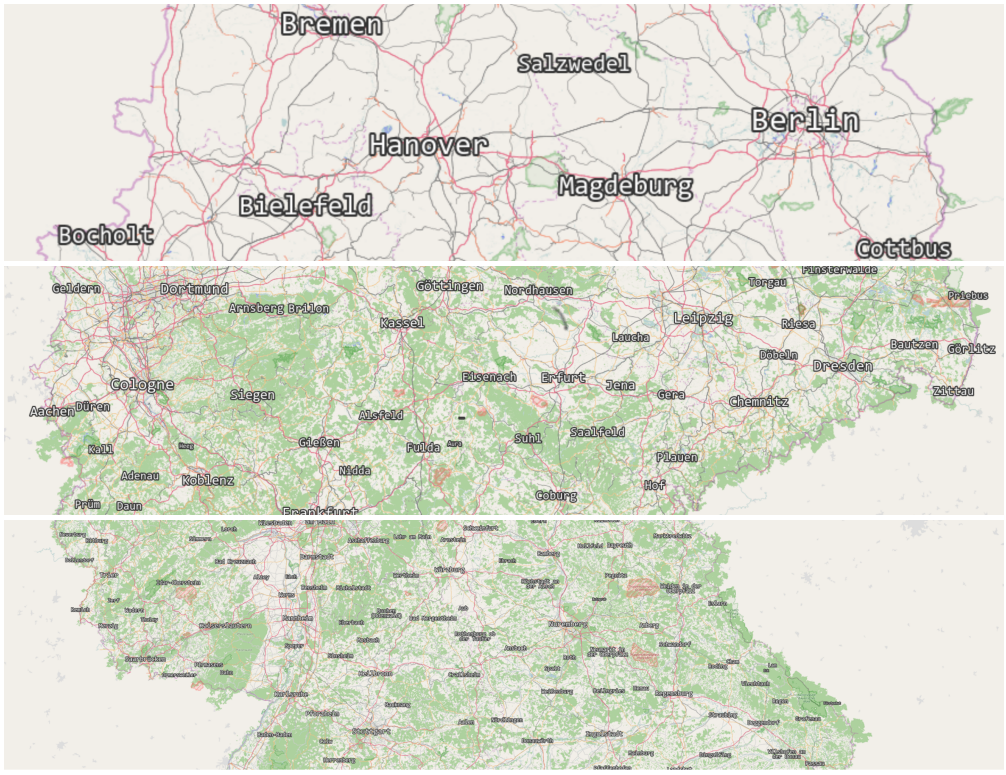

Figure 1.1.: Germany labeled in 3 different levels of detail at a coarse, medium and high level of detail (© OpenStreetMap contributors).

while densely populated regions are crammend with labels. We want the labels to be evenly distributed in the space. What brings us to the so called ”relative importance” of a label. It means that a village in a sparsely populated region might be of higher relative importance than a town next to a mega-city. You can find an example of a labeling in figure 1.1. It shows the labeling of Germany in three different levels of detail from detailed (bottom) to less detailed (top). In each level you see the labels being evenly distributed in the available space.

As we have seen, when continuously zooming out a map view, labels might get into conflict when they start overlapping. We could solve this conflict by eliminating one of the involved labels. Each time a label is eliminated during a map zoom this might recreate space for an already eliminated label. But allowing the later label to reappear might lead to flickering effects during a continuous zooming where labels vanish just to reappear on further zooming. This effect is not intended and may lead to distraction of the user. In figure 1.2 you see an example of Bing Maps where the label of Oberaichen is visible on a zoomed out view (left). When zooming in the label is removed (middle) and reinserted after

further zooming in (right).

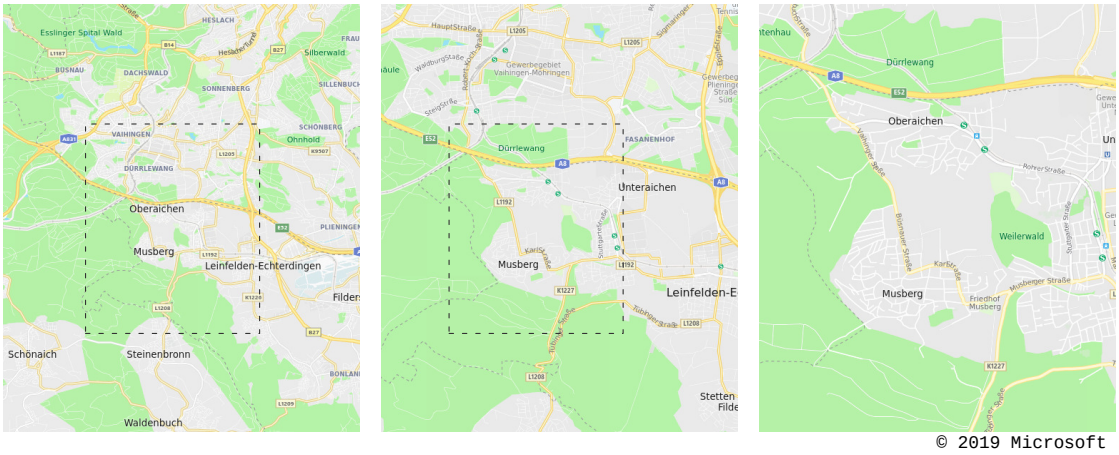

Figure 1.2.: An example of an inconsistent labeling in Bing Maps where the Oberaichen label is present at a zoomed out view (left), zoomed in the label is hidden (mid) while further zoomed in, the label pops up again (right).

Considering map rotation reveals a very similar problem. Think of the scenario where you are driving a car and your navigation system is guiding your way. You see the map and the labels on it while approaching a crossing where you have to turn left. You are now focused on driving and successfully passing the crossing. But after finishing the maneuver your map has turned and you are facing a completely different map view. What you see is a rotated map view which is still oriented in your direction of travel. With current map labeling algorithms the whole scenery would have changed - especially the displayed labels. This is because after rotating the map, some of the former labels would have overlapped after the rotation. The common way to overcome this is to hide one of the labels to resolve the conflicts. At the same time, labels which were hidden before because of overlappings, could have shown up on the screen. Again, on a continuous rotation, for example, when following a long curve, flickering of the displayed labels might occur. This could be distracting and should therefore be avoided.

The overall goal of the work at hand was to find efficient algorithms and methods to label digital maps. The labeling scheme shall be applicable to interactive maps which allow to continuously pan, rotate and zoom the map view. The problems as described in the paragraphs above should be avoided.

Many of the results presented in the work at hand were previously published at conferences. In 2016, an efficient algorithm for computing elimination sequences in 2-dimensional space was presented at the 27th *International Workshop on Combinatorial Algorithms* (IWOCA 2016) in Helsinki [FKS16]. This paper was joint work

with my supervisor Stefan Funke and Sabine Storandt. A year later a more generalized approach for general d-dimensional instances was presented at the *Meeting on Algorithm Engineering and Experiments* (ALENEX17) in Barcelona [Bah+17]. This paper also contains algorithm for elimination sequences on the sphere with a spatial data structure based on Delaunay Graphs. The paper was joint work with Daniel Bahrdt, Michael Becher, Stefan Funke, André Nusser, Martin Seybold and Sabine Storandt. The general idea for labeling points was published at the 10th *International Conference on Geographic Information Science* (GIScience 2018) in Melbourne [Kru18].

The scheme for efficiently computing area label positions is joint work with Thomas Mendel. The research has been published in [KM20].

Structure

The work at hand is subdivided into several parts, each having several chapters as their main subdivisions. The current introductory part is about introducing the topic. The second part introduces our labeling model. It is followed by a part about solving underlying algorithmic problems induced by our labeling model.

The remaining part of this first introductory part contains two more chapters: We will review relevant related work concerning the map labeling problem in the following chapter 2. The following Chapter 3 gives a short overview of fundamental primitives, concepts and algorithms.

Part II is all about map labeling. A model for labeling areas is introduced in chapter 4. A second model for labeling points of interest is described in chapter 5. For both, we define requirements for a good labeling and introduce our model. We further show how to link the labels and map scale in an interactive map setting which allows for panning, rotating and zooming the map view. In chapter 6 we describe how we retrieved label data from the OpenStreetMap data set.

The introduced labeling models involve some challenges which are interesting from an algorithmic view. In part III we focus on these problems. In chapter 4 we introduce our developed efficient algorithm for the placement of a curved box of maximum size in a polygonal area which underlies the area labeling problem. The problem of computing elimination sequences, underlying the point of interest labeling problem, is targeted in Chapter 8. There we consider the generalized problem of growing hyperballs in a d-dimensional space. We prove that the problem in general is hard to solve and introduce a simplified scenario. An algorithm is introduce to efficiently compute an optimal solution in theory. For the map labeling problem, we are considering the special case of elimination sequences on a spherical surface. Some heuristics and an LP are provided to solve the unsimplified problem.

A final conclusion is given in part IV.

Acknowledgement

Working on a project like the one presented here is not possible without the support of others. First of all, I would like to thank my supervisor Stefan Funke. He gave me my freedom to pursue the project and to push it in the direction I was most interested in. Nevertheless, he was always there when I needed advice or feedback.

The best ideas and solutions to problems are found when talking to others. I want to thank my colleagues Daniel Bahrdt, Florian Barth, Thomas Mendel, Claudius Proissl, Tobias Rupp, and Martin Seybold. It was a pleasure to discuss problems together and find solutions to them and find new ones.

My deepest thanks go to Daniel Bahrdt, Florian Barth, Tobias Rupp and my little sister Linnéa Krumpe. They all did an amazing job providing so much comments and improvement proposals to the work at hand. Without you, I would probably still be trying to find and fix mistakes and would probably never finish that.

Last but not least, I would like to thank my girlfriend Anna and my whole family for supporting me on the long way through this project.

One of the biggest drawbacks a project like this will always have, is the availability of data to deal with. Especially when compared to the big tech competitors, like Apple, Bing, Google, Here etc. As this is one of the most valuable things the companies have, one will never be able to get access to these treasures. Fortunately there is an open source project collecting geographic data from volunteers all over the world. Our deepest thanks go to all the OpenStreetMap contributors [Con17c] for doing a fantastic work creating a free and open and versatile collection of geographic data. Although the OpenStreetMap project is awesome the huge amount of data can be overwhelming. Special thanks goes to the Geofabrik [Gmb18] for providing smaller but complete pieces of the whole data set. Thanks also go to all the developers of the various tools in the OpenStreetMap ecosystem which are open source and free to use.

2. Related Work

Summarizing the research area of map labeling and especially interactive map labeling of point and area features is a challenging task. A short view into Alexander Wolff's map labeling bibliography [Wol] shows the overwhelming amount of research in this field. Nevertheless, we will point out the most important work related to the topics we are focussing on in the work at hand.

In the early 60s, it was the Swiss professor Eduard Imhof who first defined what a good map labeling looks like. This work was published in German [Imh62] and republished in English in the mid seventies [Imh75]. His research is fundamental to the whole research area and defines the very basic principles and goals of a map labeling. In the same decade, Pinhas Yoeli, professor in Tel Aviv published a first attempt of an algorithm to semi-automatically create map labelings [Yoe72]. With this pioneering research, a large research area and software market was created.

Amongst others in the 80s and early 90s, a group around Herold A. Freeman pushed the research with several publications [AFL84; Fre85; FA87] (amongst many others). In his paper [Fre86], Freeman gives a very good overview of the early works in this field of research. The Freeman group dealt with area-, line- and point labels and developed an expert system to label maps [FA87; DF92]. First attempts on using polygon skeletons to compute area label positions were made [AFL84; Fre85; Ved94]. These are early successors of our approach on area labeling. A summary of the work of the Freeman group was given in [Fre05; Fre07].

In the late 80s, the research field began to differentiate. Concerning area labeling, Arunaa A. Vedula, a student of Freeman, in his thesis [Ved94] provided a work which was solely related to area labeling. This work covered labels fitted into the area, mimicing the area shape, as well as external labels which are connected to the area with a leader. The later is a short line, pointing from the label to the corresponding area if the area is to small to place the label within. Freeman integrated this work into a more general approach [Fre95]. Pinto, Freeman in 1996 aimed for integrating all the existing solutions for special cases of the area labeling problem into one approach [PF96]. The introduced feedback approach used different techniques to determine initial placements which are then evaluated and optimized according to human feedback. In 2001, Mathieu Barrault formalized a measure of an area label quality [Mat01]. Based on this measure he developed an approach to fully automatically label areas based on circular approximations of the area skeleton. His work can be considered the direct predecessor of the

area labeling approach at hand. An alternative approach for area labeling quality assessment was presented by Steven F. van Dijk et al. in [Dij+02].

In the field of point labeling, the first research aimed for automatically placing the point label to maximize the number of labels which can be displayed. This was targeted for example by Robert G. Cromley [Cro85] who used linear programming techniques. Michael Formann and Frank Wagner [FW91] implemented a 2-approximation algorithm to approach an optimal solution. They also showed that the point labeling problem with four possible label positions is NP-complete. This was also shown by Joe Marks and Stuart Schieber [MS91] in the same year, 1991. Jon Christensen, Shieber and Marks used gradient descent and simulated annealing techniques for solving the point labeling problem in an eight possible position setting [CMS95]. A second challenge in this field of research was the following: Assuming there are more points we want to label than space available, which points should be selected and which ignored? Douglas M. Flewing and Max J. Egenhofer further pushed this issue and tried to formalize the settlement selection problem [FE93].

Until then, the presented solutions aimed for supporting the creation of static maps, e.g. paper maps. In the late 90s, the question of the labeling of maps in a dynamic setting arose. Marc van Kreveld, Rene van Oostrum and Jack Snoeyink. in 1997 [KOS97] built on an idea, Langran and Poiker presented in [LP86]. They developed a ranking of the points which allows to retrieve a map labeling without extensive computation at runtime. Their work already introduced the concept of monotonicity of the labeling during zoom. These two ideas make it a early predecessor of our work presented in the paper at hand. Ken Been, Eli Daiches and Chee Yap in 2006 and Been, Martin Nöllenburg, Sheung-Hung Poon and Alexander Wolff in 2010 developed this idea further by linking the computed ranking directly to the map scale [BDY06; Bee+10]. Their precomputed active ranges allow to determine a labeling of a map of arbitrary zoom level with one simple filter step. In [BDY06], they introduced a set of general consistency desiderata to be fulfilled by a labeling scheme for interactive maps. Yet, they confined their approach to interactive maps allowing only panning and map zoom - not rotation. Nadine Schwartges, Dennis Allerkamp, Jan-Henrik Haunert and Wolff in [Sch+13] used a linear program formulation and some heuristics to compute and approximate active ranges for disks with unique growth factor.

Andreas Gemsa, Nöllenburg and Ignaz Rutter in [GNR11; GNR16] addressed the problem of labeling rotating maps. Their approach is to compute one or several rotation angle ranges for each label where it can be visualized without overlaps. But they did not consider map zoom in their work.

3. Preliminaries

We will now introduce some basic primitives, concepts and algorithms to establish a common knowledge with the reader.

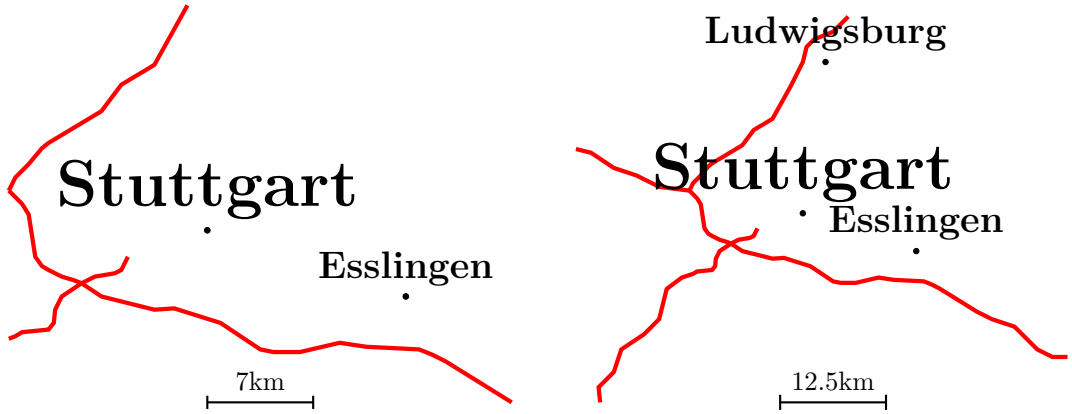

Figure 3.1.: Schematic map of the Stuttgart region at two map scales. A larger map scale left and a more zoomed out view with a smaller map scale (right).

Since we are mainly talking about map and map visualization, we will first eliminate the main source of misunderstanding. This is about what it means to have a small and large map scale and what zooming in and out of a map view means. Let us consider a map like the one illustrated in Figure 3.1. On the left there is a map showing the surroundings of Stuttgart at a specific map scale, say $1cm$ is equals to $7km$ (may depend on your print of this paper though). The map scale is $1 : 700,000$ then. If you **zoom out** a bit, you get the map shown on the right hand side. There the map scale changed to $1 : 1,250,000$ - a **smaller** map scale (consider the result of the division $1/700,000$ and $1/1,250,000$). So a map view showing whole Europe has a small map scale while a zoomed in view showing a single city is of a much larger map scale.

When zooming out of a map, distances in the map view are shrinking. See the Stuttgart - Esslingen distance in Figure 3.1 from left to right for example. When keeping labels at a constant size, they are moving closer together, eventually leading to labels overlapping each other. This is the main source of trouble when dealing with labelings of interactive maps allowing continuous map zoom.

In Figure 3.2, you see the same map view in standard orientation (north facing)

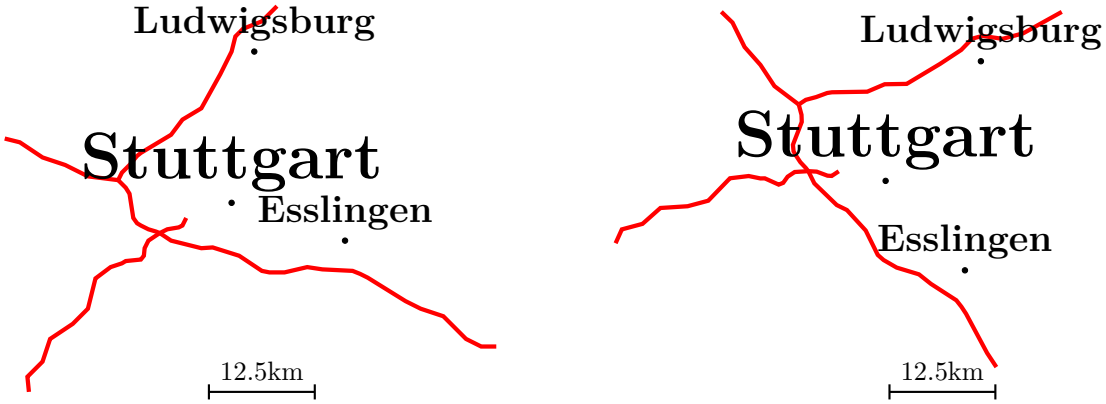

Figure 3.2.: Schematic map of the Stuttgart region. The map is oriented north on the left and rotated by −30 degrees on the right hand side.

on the left and rotated by 30 degrees in clockwise direction. Since the lettering normally remains vertically aligned, its orientation related to the map changes on rotation. You can easily see that rotating the map view in counterclockwise direction would have led to the labels of Stuttgart and Esslingen overlapping each other. This is the main challenge when looking at labelings of maps which allow for rotation.

For a given map setting, we can check if two label overlap by computing the size of the label's bounding boxes. By using the current map scale we can transform this box size to geographic distance, like meters. This allows us to determine whether two labels will overlap in a setting or not. Conversely, it allows us to determine the largest scale at which two labels start overlapping.

3.1. Points, Lines and Areas

The features we are about to label on a map can be classified into three different types: point, lines and areas. A **point** is defined by 2 coordinates in a 2-dimensional space. This can easily be generalized to d dimensions. The coordinates define the location relative to the origin of the coordinate system in the two dimensions (x and y in the example depicted in Figure 3.3) In the widely used WGS84 coordinate system, a unique position on the earth's sphere is defined by its latitudinal and longitudinal angle [Wik19d]. The coordinate system is located at the center of mass of the earth. The latitude value describes the angle to the equator. The longitude value defines the one to the Greenwich meridian.

A sequence of several consecutive points can be used to define the course of a **line**. See the center of Figure 3.3 for example. Since such a line is composed of

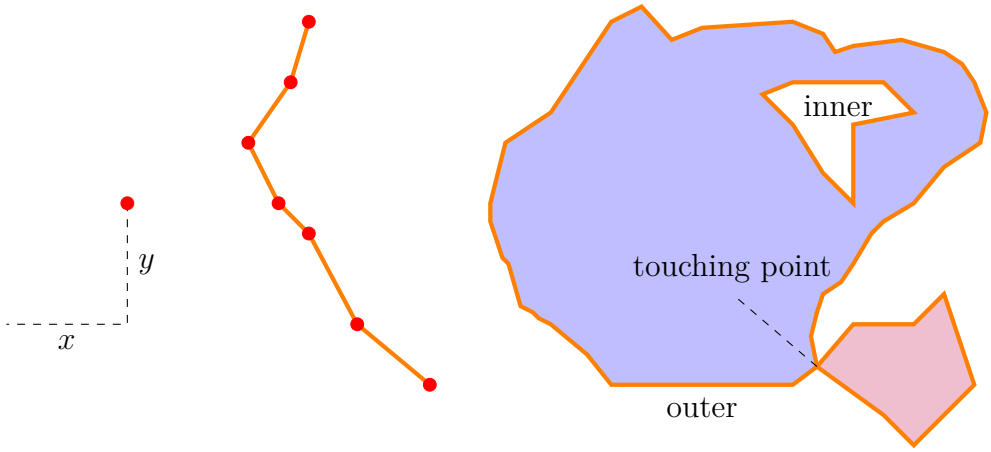

Figure 3.3.: The fundamental primitives: a point (left), a polyline defined by a sequence of points (mid) and a polygon with a hole defined by an outer and inner polyline (right).

several line segments we also call this a **polyline**. A line segment (or just segment) is defined as a line segment between two given endpoints. Polylines can be used to describe streets, borders and the like in a geographic setting.

A closed polyline (also called polygon), where the first point is equal to the last point, can be used to define an **area**. More complicated areas can be specified by an outer polygon defining its shape while inner polygons define holes of the area. See Figure 3.3 right for example.

For the sake of simplicity, we suppose well-formed data in the following. It means we assume points to be in general position i.e. no two points share the same coordinates. This can easily be guaranteed by slightly perturbing the data if necessary. We further assume polylines not to intersect each other and itself. If so, we may insert the intersection point into one or both polylines. For an area we suppose that holes are completely contained within the area. Additionally, for the outer boundary we suppose that it does not have any self intersections nor touching points where the boundary touches itself. An example is illustrated in Figure 3.3. Such a situation may be resolved by splitting the area into two (blue and purple in the drawing) at the touching point.

3.1.1. Computing Distances of Geographic Features

Computing distances between two geographic features is not trivial even between two single point-features. Since they both are located on the earth surface, the distance between must be measured along the sphere surface. Therefore we use the greater circle through the two points with its center being the sphere center.

The arc length between the two points is defining the distance between them. See Figure 3.4 for an illustration. This particular distance can be computed using the Haversine formula [Wik19c]. Due to some trigonometric computations involved this is quite expensive to compute. In the work at hand, when we refer to the distance $|p, q|$ between two points p and q, this greater circle distance is assumed if not otherwise specified.

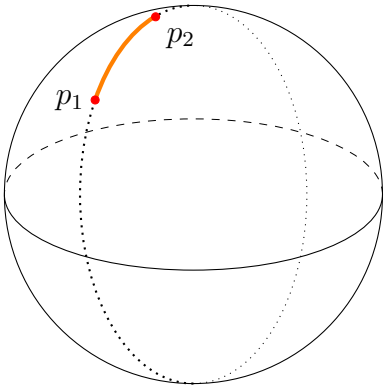

Figure 3.4.: The distance between two points p_1 and p_2 on a sphere is the length of the circular arc of the greater circle through p_1, p_2.

We introduced this notion of distance between two simple points only. Computing distances with line or area-features involved, induces several other problems. But since we only need the distance between two points, we will not deal with them here.

3.2. Fundamental Concepts

Having defined points, areas and point distance, we may now proceed to more sophisticated, derived concepts.

3.2.1. Delaunay Triangulation and the Voronoi Diagram

For a given point set in 2-dimensional space, we can compute a triangulation. This is a subdivision of the convex hull of the point set into triangles where the points are the triangle corners.

In Figure 3.5, you see a triangulation depicted for a point set of 5 points $\{p, q, r, s, t\}$. This particular triangulation is a special one. If you look on each triangle and its circumcircle, this circumcircle does not contain any other point from the point set. Such a triangulation is called a **Delaunay triangulation** [Kre+00]. This **Delaunay property** prevents triangles with small angles to be

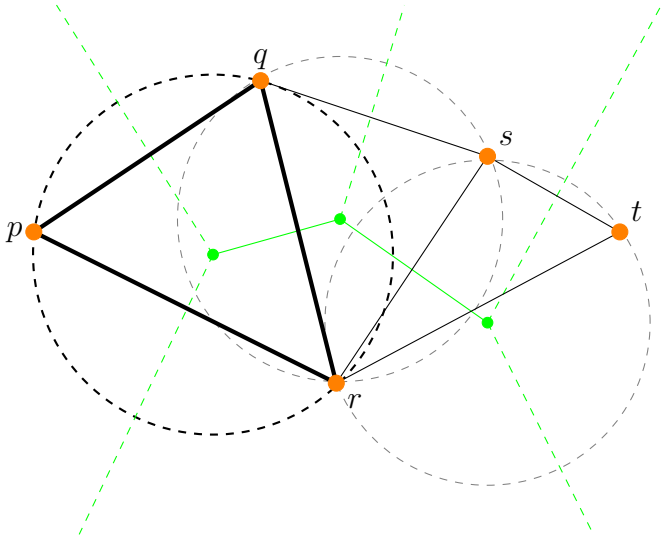

Figure 3.5.: A point set in 2d. The Delaunay triangulation (black edges) with the corresponding circumcircles (dashed). The Voronoi edges, separating the Voronoi cells are drawn in light green. The Voronoi vertices (green dots), between which the edges are drawn solid.

contained in the triangulation. So in this particular example, it ensures that the $q - r$ edge is used instead of the $p - s$ edge to triangulate the p, q, r, s subpart. In this way it maximizes the minimum angle in the triangulation [Kre+00]. Which leads to the effect that long and thin triangles are suppressed for the benefit of triangles of a more uniform shape. You can find an example in the above Figure 3.5. Replacing the $q - r$ edge with the $p - s$ edge induces the p, q, s triangle with a much thinner, non-uniform shape.

While the delaunay triangulation subdivides the space into triangles, the Voronoi diagram partitions the space according to nearest neighbor relations. Let us start with a point set as before and consider Euclidean distances. For each point p in the point set there is a cell containing the points in space, for which p is the nearest point in the point set. This is called the **Voronoi diagram** of the point set [Kre+00]. The cells are actually intersections of halfspaces. These intersections, the so called Voronoi vertices, and the bisectors between them, define a graph. This graph is called **Voronoi graph** or sometimes also called **Voronoi diagram**.

The Voronoi diagram turns out to be dual to the Delaunay triangulation. The Voronoi vertices are equal to the center points of the Delaunay circumcircles. Connecting the center points of the circumcircles whenever the corresponding delaunay triangles share an edge, gives the Voronoi edges [Kre+00]. In Figure 3.5, you see the Voronoi vertices depicted as light green dots and its edges in green. The dashed edges are infinite edges, bounding the infinite Voronoi cells at the outside of the

Voronoi diagram.

3.2.2. The Medial Axis

The Voronoi diagram as defined above can be used to approximate the solution to the following question: Considering an area, one might want to find something characteristic describing the area.

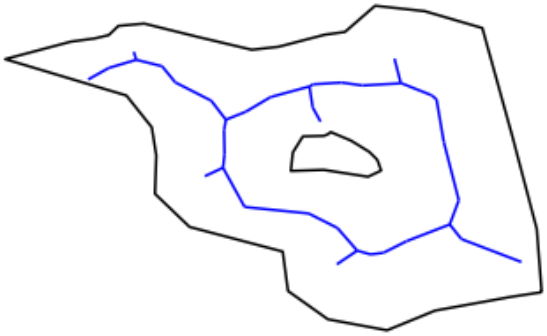

Figure 3.6.: Approximation of the medial axis of a polygon using a subset of the Voronoi edges.

The **medial axis** of a planar shape was first defined by Blum in 1967 [Blu67] and Lee in 1982 [Lee82] amongst others. Given a simple polygon P representing the shape. Its **medial axis** is defined as the locus of points p internal to P such that at least two points on the polygon's boundary are equidistant and closest to p. This definition can be applied to polygons containing holes in a straightforward manner. Each point on the medial axis can be assigned a radius, describing the distance to the boundary [DZ04].

As Schmitt in [Sch89] and Brandt in [Bra94] point out, the medial axis, also called skeleton, of a polygon can be approximated using Voronoi diagrams. A special subset of Voronoi edges, namely those who are completely contained in the polygon, are an approximation to the medial axis [MS00]. You see an example depicted in Figure 3.6.

3.2.3. Spatial Relations of Point Features

Considering a set of geographic features - for the sake of simplicity we restrict to point features here - one might want to know about their spatial relation. In the following work, we are interested especially in two relations: the **nearest neighbor** of, and the set of all **neighbors in a specific range** around a given query point. Please see Figure 3.7 for an illustration of the concepts we introduce in the following.

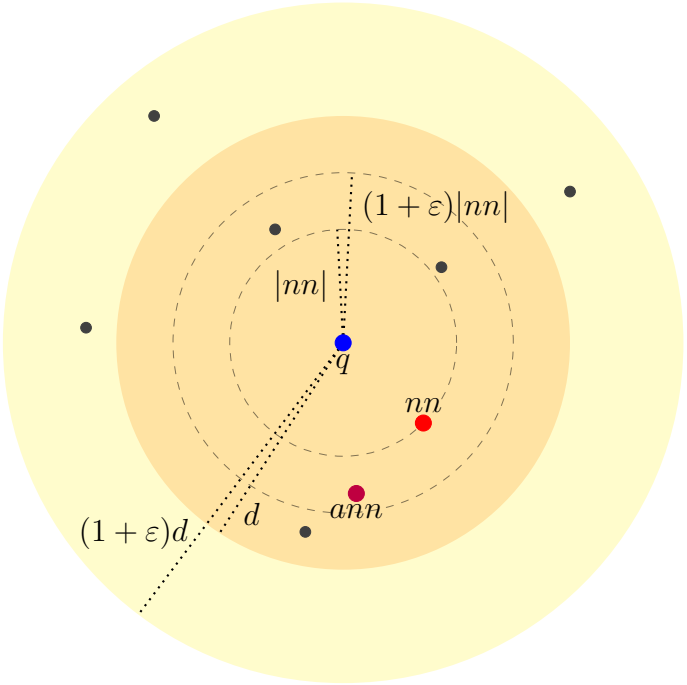

Figure 3.7.: A query point q (blue) with its nearest neighbor nn (red) and an approximate nearest neighbor ann (dark red). A range contains all points contained in the dark yellow area. The approximate range in yellow contains all points from the range and may contain some of the light yellow area.

We will first introduce the concept of a nearest neighbor and, slightly weakened, an approximate nearest neighbor. We define the two as follows:

Definition 1 (NearestNeighbor)**.** *For a point set $P \subset \mathbb{R}^d$ and a query point $q \in P$, the nearest neighbor of q is a point $p \in P \setminus \{q\}$ with $|p,q| \leq |p',q|$, for all $p' \in P \setminus \{q\}$.*

Definition 2 (ApproxNearestNeighbor)**.** *For a point set $P \subset \mathbb{R}^d$ and a query point $q \in P$, an ε-approximate nearest neighbor of q is a point $p \in P \setminus \{q\}$ with $|q,p| \leq (1+\varepsilon) \cdot |q,p'|$, for all $p' \in P \setminus \{q\}$.*

So for a set of points in d-dimensional space, a **nearest neighbor** of a query point is the point in the point set with minimum distance to the query point. Obviously this nearest neighbor must not be the query point itself.

Slightly weakening the condition leads us to the definition of an **approximate nearest neighbor**. For a query point and a parameter ε, we search for a point whose distance does not deviate much from the distance of the real nearest neighbor. In fact, we want the distance difference to be less than ε-times the real nearest

neighbor distance, i.e. the distance of the approximate nearest neighbor must be $< (1 + \varepsilon)$ times the nearest neighbor distance. Given in simple words, for the reported point its distance to the query point is at most ε times larger than the true nearest neighbor distance. In the above Figure 3.7 you find an example of a query point q - the blue dot. Its exact nearest neighbor is the bright red *nn*. The distance between q and the dark red *ann* is about 1.4 times the nearest neighbor distance. Hence for an $\varepsilon > 0.4$, *ann* may be reported as an approximate nearest neighbor of q.

Let's introduce the concept of a range, or a neighborhood, and is approximate counterpart in the following.

Definition 3 (Range)**.** *For a point set $P \subset \mathbb{R}^d$, a query point $q \in P$, and a distance r, the range of distance r around q is the set $S = \{p \in P \setminus \{q\} : |q, p| \leq r\}$.*

Definition 4 (ApproxRange)**.** *For a point set $p \subset \mathbb{R}^d$, a query point $q \in P$, and a distance r, an ε-approximate range reporting query returns a set S with $S \supseteq \{p \in P : |q, p| \leq r\}$ and $S \subseteq \{p \in P \setminus \{q\} : |q, p| \leq (1 + \varepsilon)r\}$.*

For a set of points in a d-dimensional space, we define the **range** around a query point q and a distance r to be the subset of points whose distance to q is less or equals than r. Similar to the approximate nearest neighbor, we define the **approximate range** to contain points whose distance to the query point does not deviate more than $\varepsilon \cdot r$ from r, i.e. their distance to the query point is $\leq (1 + \varepsilon)$ times the query distance r. While we want **all** points of distance $\leq r$ to be contained in any case, points of larger distance may or may not be contained in the result set.

3.3. Fundamental Algorithms

3.3.1. Computing Delaunay Triangulations

In Section 3.2.1 we defined the Delaunay triangulation. We will now briefly introduce an algorithm to efficiently compute such a triangulation. The algorithm is randomized and incremental. It iteratively inserts the points of the point set and maintains the Delaunay property thereby. Please see [Kre+00] for more details.

In the following, let $P = \{p_1, p_2, \ldots, p_n\}$ be a point set in $2d$ of the size n. For the sake of simplicity, we assume general position. Which in this case means that no two points are located at the same position, no three points are colinear and no four points are cocircular. Starting with the rightmost point p_1, we introduce two auxiliary points p_{-1}, p_{-2} such that the whole point set P is located in the triangle $p_1 p_{-1} p_{-2}$. This single triangle is a valid Delaunay triangulation of the point set

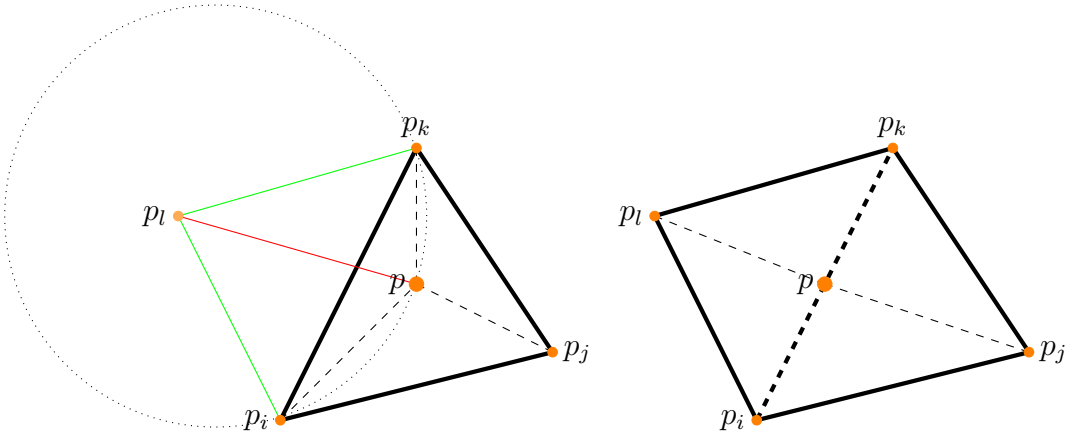

Figure 3.8.: Inserting a point p into Delaunay triangulations. If p is located within a triangle (left), add new edges to each corner of the triangle. If p is located on an edge, split the edge and add new edges to the remaining vertices of the two adjacent triangles. If the Delaunay property is violated for one of the created triangles (left: p_ip_kp), we flip the edge $\overline{p_ip_k}$ to $\overline{p_lp}$. Which might invalidate the edges $\overline{p_kp_l}$ or $\overline{p_ip_l}$ which we have to fix recursively etc. .

$\{p_{-2}, p_{-1}, p_1\}$. We compute a random permutation of $P \setminus \{p_1\}$ and insert them into the given permutation iteratively.

When inserting a point p two things can happen: If p is located within a triangle $p_ip_jp_k$, we add new edges $\overline{p_ip}$, $\overline{p_jp}$ and $\overline{p_kp}$ to the triangulation (see Figure 3.8 left). Then for every edge of the original triangle we consider the triangle p, q, r with p being our inserted point and $q \neq r \in \{p_i, p_j, p_k\}$. Let s be the point defining the other adjacent triangle of the edge $\overline{qr}$. We check if the edge is legal by checking if the two circumcircles are containing the corresponding other point. If not, we are done. Otherwise, we flip the $\overline{qr}$ edge, i.e. replace it by $\overline{ps}$. Since this could have made the edge $\overline{qs}$ and/or $\overline{rs}$ illegal, we continue checking these two edges and flipping them if required.

In the second case p, is located on an edge connecting two points p_i, p_j (see Figure 3.8 right). Let p_k, p_l be the two points defining the two triangles adjacent to the edge $\overline{p_ip_j}$. Then add p by introducing edges to all four points p_i, p_j, p_k, p_l and legalizing the remaining adjacent edges.

This algorithm proves to be correct [Kre+00]. It further turns out to compute the Delaunay triangulation in expected time $\mathcal{O}(n \log n)$ using $\mathcal{O}(n)$ expected storage.

3.3.2. Answering Spatial Queries using Delaunay Triangulations

The previously introduced Delaunay triangulation can be used to answer queries for the NearestNeighbor or the Range in a point set. In the following let P be the point set in a 2-dimensional, euclidean space. Let's suppose that a Delaunay triangulation is given for P.

Since the nearest neighbor graph is a subgraph of a Delaunay triangulation, for a query point $q \in P$, its NearestNeighbor in P is a direct neighbor. Hence in order to find the nearest neighbor we just have to inspect the edges incident to q.

Given a query point $q \in P$ and a query distance r the Range, i.e. finding all points in P with a distance $< r$ to q, can be done by a simple graph traversal. Starting at q we visit every adjacent vertex and check if its distance to q is less than the query distance. If so, we report the point and recursively continue at that point. If not, we skip the point and continue.

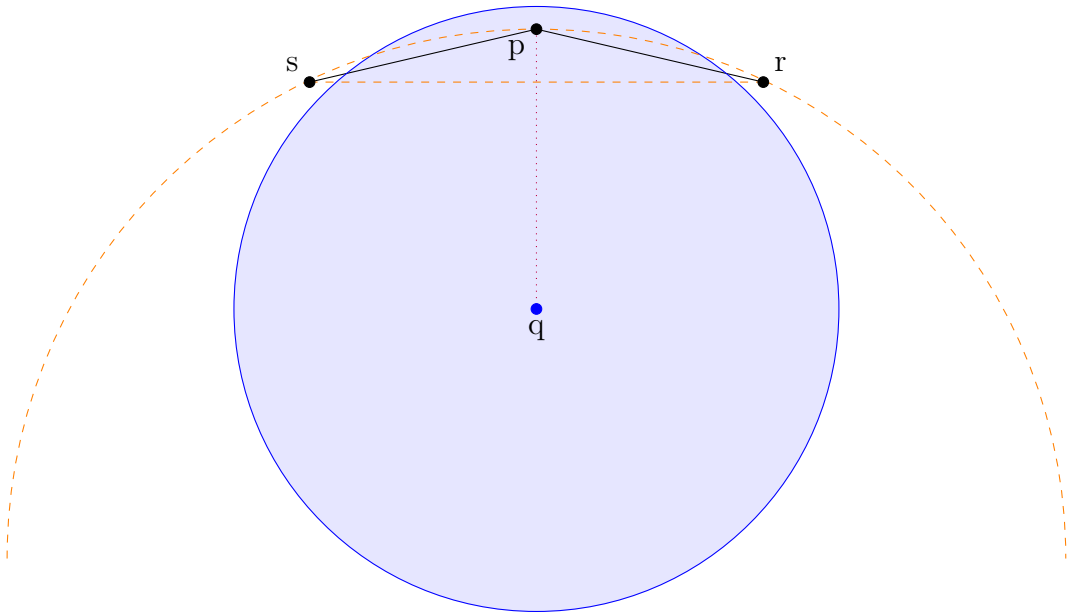

Figure 3.9.: Supposing p is not connected to q via an path contained in the Range of q. Then p, s, r form a triangle whose circumcircle contains q contradicting the Delaunay property.

This procedure is correct, since the following holds: If p is contained in the Range of q, then p is directly connected to a point v in Range. Since a triangulation has only one connected component, this suffices to prove that there is a path $p, \ldots, q$ and every point is contained in the Range.

For the sake of contradiction, suppose that p is not connected to a point in Range. We know that either pq are connected via an edge or there is at least one

edge intersecting the direct link between p and q (dotted red in Figure 3.9). Let s, r be the one of these edges which is closest to p on that link. Since the point set is triangulated, psr must form a triangle and $s, r \notin \mathbb{R}$. Hence the circumcircle defined by p, s, r needs to have a larger radius than the query distance r. Thus it contains the query point q. The latter contradicts the Delaunay property. Hence proving our assumption to be wrong.

3.3.3. Linear Programs

Even though it is more a concept for problem modelling, linear programs allow to compute solutions to so modeled problems. Since many problems, even complex ones, can be modeled by linear programs, we will introduce the concept. It will later be used to compute optimal solutions to problems where we only give approximation algorithms. Being able to determine optimal solutions to at least smaller instances of such problems will allow us to bound the quality of our approximation.

A linear program is built from essentially three components: A set of **variables**, an **objective function** which should be maximized or minimized. The third component is a **constraints** set. These constraints determine the feasible variable values from which we try to find the ones optimizing our objective.

A small example is the following example which determines two values x and y such that x is twice as large as $y + 5$. Both values are larger than or equals to 0. The sum of the two is minimal:

$$\begin{array}{lrcrl} \min & x & + & y & \\ s.t. & x & - & 2y & = 5 \\ & x & & & \geq 0 \\ & & & y & \geq 0 \end{array}$$

What makes such a program a **linear program** is the fact that each constraint and the objective function are **linear** in the variables. Which means that no two variables are multiplied with each other and no variable has an exponent unequal to 1.

Having a valid program for a given problem allows us to find an optimal solution (if it exists) using standard techniques and solvers. More detailed information is provided in [Kre+00] and [WS]. A well known solver is the one provided by Gurobi [Gur19].

Linear programs can be categorized based on the possible values. So we distinguish between LPs where the values may have arbitrary real values. They can be solved deterministically in polynomial time using the ellipsoid method [WS]. In general more difficult to solve are ILPs (Integer Linear Programs). They are similar to general LPs with the difference that the possible variable values are re-

stricted to integral values. Solving these is NP-complete [Pap81]. For more details see [Kre+00].

Part II.

Map Labeling

In this part, we will address the labeling of interactive maps. For us, ”interactive” means that the map view can be panned, rotated and zoomed. These interactions may be because of an user interactively exploring the map. Alternatively, in a navigation system, the view may be adjusted automatically to match the users direction or to provide a more or less detailed view depending on the current situation.

Figure 3.10.: Labeling of a political map of europe with area labels computed by the introduced method.

In the work at hand, we propose a labeling scheme for areas and point of interests. The following Chapter 4 introduces the labeling model for areas. We show how the area labels and the map scale can be combined to form a consistent view. An example for this area labeling approach is illustrated in Figure 3.10. The subsequent Chapter 5 describes our point labeling scheme. We define criteria for good labelings, define our model and discuss some extensions to the model. The later can help to increase the number of labeled objects. In the last chapter of this part, we will quickly have a look at the question where to get data sets from (Chapter 6).

Imhof in 1962 (see [Imh62] and [Imh75]) outlined a set of general principles and requirements for good label placement:

1. *Legibility*, which means that labels should ”be easily read, easily discriminated and easily and quickly located.” [Imh75]

2. *Clear graphic association* strives for unambiguous identification of the object the label belongs to.

3. Map labels should avoid covering, overlapping and concealing other map content.

4. Map labels should ”assist directly in revealing spatial situation, territorial extent, connections, importance and differentiation of objects.” [Imh75]

5. Variation of label style and size should be used to visualize the classification and hierarchy of the labeled map objects.

6. ”Names should not be evenly dispersed over the map, nor should names be densely clustered” [Imh75]

These principles are the basis of nearly every work which deals with labeling maps. They will lead what we are doing in the following pages. The interpretation of the principles strongly depends on the context, i.e. which kind of labels we are considering. Therefore we will discuss them in detail at the beginning of the corresponding chapters.

4. Labeling Areas

Area label may be applied to every type of areal feature, may it be administrative areas, lakes, islands, woods and the like. The only precondition is that the map scale is large enough to place the label within the area. If this is not the case, alternative labeling techniques can be applied like boundary labeling or other leader-based techniques, key numbering, point labeling and others (see the former Chapter 2).

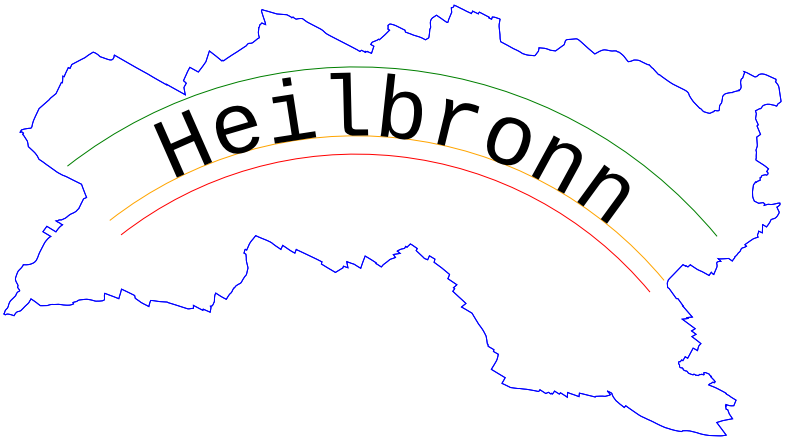

Figure 4.1.: An area label for the Heilbronn city area.

The problem we are faced with is the following. We are given an area which is defined by an *outer boundary* and might contain a set of *holes*. We aim for placing its name within the area such that it represent the corresponding region as good as possible. To reach this, the label may be bend as well as stretched to fit the area shape. See Figure 4.1 for an example.

M. Barrault in 2001 ([Mat01]) and S. van Dijk et al. in 2002 ([Dij+02]) both formalized the principles of good area labels. We will discuss the formalization defined by Barrault in detail in Section 4.1. Barrault also introduced an algorithm to compute such labels. Our algorithm is based on his work but improves several parts to reach better performance.

Parts of this Chapter 4 are joint work with Thomas Mendel. They may also be contained in his dissertation thesis [Men20]. A preliminary version of this work was published in [KM20].

Barrault is considering a slightly different problem. In his work, the label font size is fixed while the inter-letter (sl) and inter-word spacing (sw) are variable. See Figure 4.2 for an illustration of sl and sw.

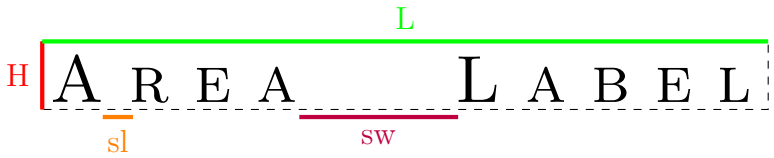

Figure 4.2.: A label with the variable spacings sl (inter-letter) and sw (inter-word) and the deduced measures L (label length) and H (label height).

We are considering the problem as follows. Let the label be an arbitrary name string with a given font type and style, e.g. bold, italic, etc. For fixed spacings sl and sw, we seek for finding a placement which allows to maximize the font size. Details are provided in Section 4.2. Finding this placement later on allows us to determine the map scale at which the label may be legible. We will discuss this in detail in Section 4.3. At small map scales, where the label within the area might be unreadable, we need to use alternative labeling techniques. These may be point labeling or leader based techniques. Both have in common that one needs a single point representing the area shape. We will discuss the problem in the last Section 4.4 of this chapter.

In the following, let us consider the placement of an area label abstractly. The reader can find an illustration of the terms and parameters in Figure 4.3. The center point and radius of a circular arc are defining the support line of the label - the circular arc along which the label is bent. The possible label position is bounded by the points where the arc is intersecting the polygon. A label position is determined by two points on this arc in between of these two intersections. All these four points can be described by angles determining the points on the support line. This concrete position is called the *baseline* of a label, green in Figure 4.3. A valid baseline length is obliviously determined by the label length and the chosen spacings. In [Mat01], the latter are variable and can freely be chosen from given ranges.

The question is, how to formalize this best fit criterion to evaluate and find an optional position?

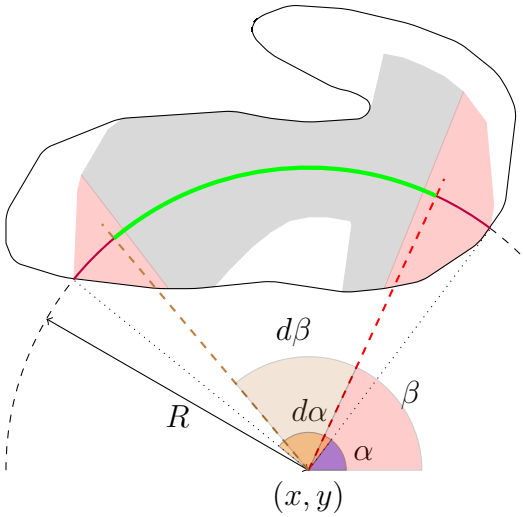

Figure 4.3.: Baseline of a label (green) determined by the support line (purple) and the angles β and $\beta + d\beta$. The perceived coverage of the corresponding labels can be computed by integrating the minimum distance to the boundary polygon for each point at the baseline. For two thirds (gray) of the support line the distances contribute positively, while the first and last sixth (light red) are negatively correlated.

4.1. Quality Measures for Curved Area Labels

M. Barrault is identifying 6 criteria which are likely to influence the quality of a label. In the following, *longitudinal* is used to describe the left-right dimension and respectively *latitudinal* the top-bottom dimension.

Longitudinal extent: The extent along a circular arc should be maximized.

Longitudinal centre: The label should be centered in the polygon in the longitudinal dimension …

Latitudinal centre: …as well as in the latitudinal dimension.

Conformity: The base arc of the label should be conform to the shape of the labeled area.

Orientation: The more horizontally the label, the better.

Curvature: A label based on an arc with larger radius is preferred.

From these criteria, Barrault is deriving what he calls the *perceived coverage* of the polygon by the label. The circle center point and an arbitrary point of the

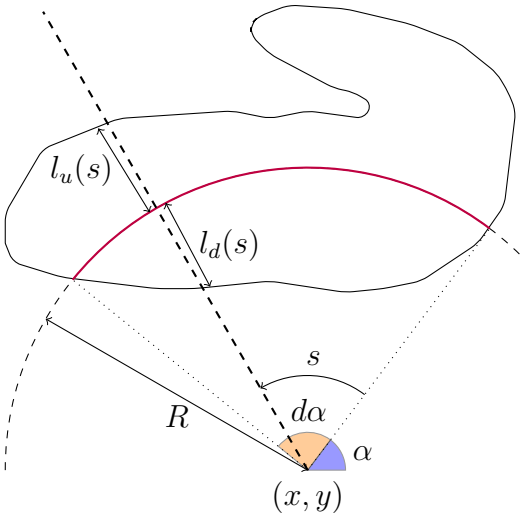

Figure 4.4.: For each point s of the support line (depicted in purple) the upper space $l_u(s)$ and lower space $l_d(s)$ determines the coverage in the polygon.

support line are determining a ray which cuts the polygon at least twice. We define the distance to the first intersection above and below to be the upper and lower space of the point (see Figure 4.4). Integrating along the baseline, summing the minimum of these two spaces, gives us the coverage of the label. However, since there should be space left before and after the label and the polygon, an additional cost is induced if the endpoints of the label are too close to the border of the area (see Figure 4.3). This is to prevent labelings from looking crammed. With this at hand, one can evaluate and compare different placements and choices of the spacings to find an optimal position along an arc.

4.2. The Labeling Model

M. Barrault is considering the problem of placing a label of fixed font size within the area. To fit the label to the area shape, he uses variable inter-letter and -word spacings. The underlying scenario is to label the area on a map with a fix map scale.

When considering dynamic maps, this focus point shifts slightly: one aims for finding a label position which allows to maximize the label extent and hence its font size. Having computed such a label position allows to determine the map scales at which the label can be visualized appropriately. Our goal is to answer the following question:

> Considering a polygonal area, what is the largest label which we can place in the area such that the label fulfilles the criteria of a good labeling as defined above.

To find this placement, we suppose the inter-word and inter-letter spacing are fixed. With this at hand, we can compute the bounding box of the label for a given font size. To meet the requirement of Imhof: to "leave a space at least one-and-one-half the size of the letters on either end of the word" [Imh75, p. 136], we adapt the label accordingly. This gives us the length L and height H (see Figure 4.2) of the bounding box of the label. From these, we can compute the ratio $A = \frac{H}{L}$, i.e. the ratio of the height to the length of a label. This ratio is independent of the actual font size.

The underlying theoretical problem of our area labeling problem now is as follows:

> What is the largest box of the given length to height aspect ratio which we can place within the area?

We have the following contraint set given:

- The box is allowed to bend along a circular arc (preferably with a large radius).
- The box should be centered in longitudinal as well as latitudinal dimension.
- It should be conform to the shape of the area
- A horizontal aligned label is preferred.

In Chapter 7, we introduce an algorithm to efficiently compute such an optimal box position and size. The algorithm is based on the idea provided by Barrault but avoids some of the drawbacks of his approach and hence achieves much better performance. So we are able to compute area labels very efficiently in near-real time (see Chapter 7 for details and timings).

With this algorithm at hand, we can think of more advanced methods to the labeling of areas in an interactive settings. Consider the scenario where an area is only partially visible, i.e. a larger part of the area is not contained within the current map view. Using fixed area labels causes problems if the label is mainly placed in the invisible part of the area (see Figure 4.5 for an example). By being able to quickly compute area labels, we can label the visible part of the area only. Alternatively, we might label a subpart of the area reaching into the invisible area. This might indicate that the area is extending in this direction, while the main part of the label is readable. The whole area and label can then be interactively

Figure 4.5.: Political map of the European countries with Russia labelled statically (left) and with a dynamic labeling (right).

explored by panning the map accordingly. An example can be found in Figure 4.5. A large part of the Kazakhstan label is visible on the left hand side. This gives a hint to the reader that the area is continuing in the non-visible part. In contrast on the right hand side, the cropped area label does not give any indication about the area extent in this invisible part.

4.3. Linking Labels and Map Scale

Having determined an optimal label position, we can derive the map scales at which the label can be displayed. Suppose we have a specified range of target font sizes, we want labels to be rendered with. Scaling the area such that the label matches the smallest and largest font size, gives us two map scales - let's call them the lower and upper map scale bounds. For this particular area, these are the map scales at which it can be labeled by the area label.

With this information at hand, we can raise the following two question:

1. Consider a continuous zooming into the map. The label of the area gets larger during the zooming. What happens if the map scale exceeds the upper map scale bound?

2. Considering a continuous zooming out of the map (the area label size shrinks in this case). What happens if the map scale goes below the lower map scale bound?

In the first case, the labeled area spans a large part of the map view. We propose to use this map scale bound to further refine the area and label its subareas - if available.

In the second case the area is to small for the inscribed label to be readable. We propose to add a point label which labels the object on smaller map scales.

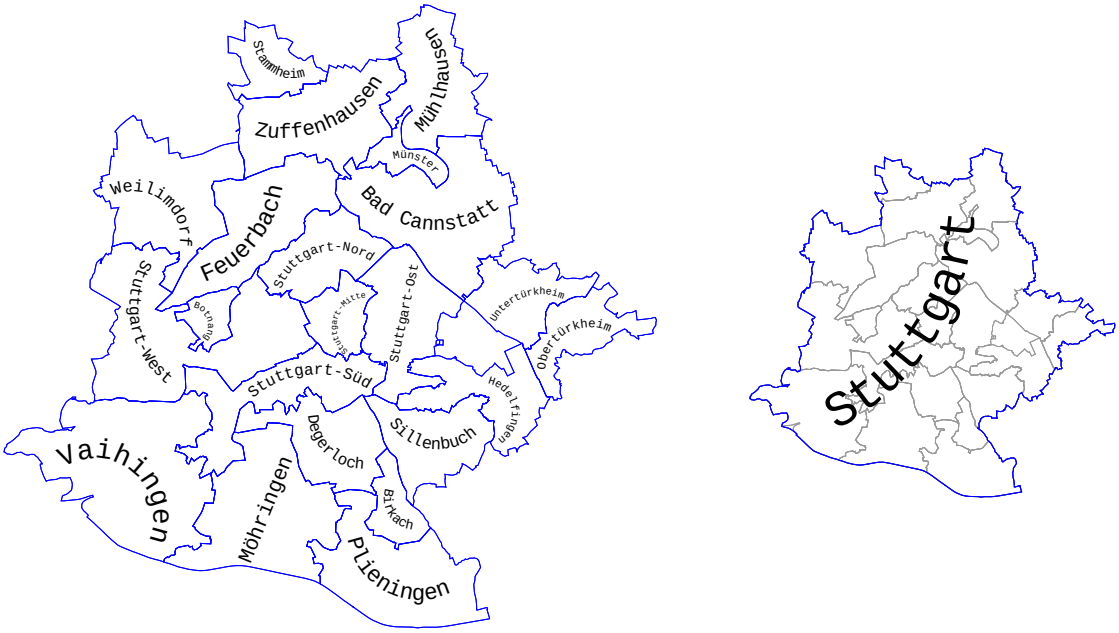

Figure 4.6.: Stuttgart with its suburbs labeled for two different map scales. On a large scale the suburbs are labeled separately (left) while on a small scale the city area is labeled as a whole (right):

Additionally, this lower map scale bound indicates that the area might be replaced by a larger, enclosing entity. As an example consider the city districts of Stuttgart which might all be labeled separately in a given map scale range. On smaller map scales the city district labels are replaced by a label of the city itself. The reader may find an illustration in Figure 4.6.

Refining and labeling the subareas separately works well if the subareas can all be labelled with similar sizes. Unfortunately, in real world data some of the refined areas might be to small to be labelled with an area label themselves. In these cases, a point label can be used to represent the area. In the following section we will discuss how to find an anchor point for the label which represents the polygon and integrates well to the zooming process.

4.4. Finding Representative Points for Areas

To find an anchor point of an associated label for an area is not a trivial question. The problem we are faced with is the following. For a given area we want to find an appropriate anchor point for a point label. This anchor point shall allow a clear attribution of a point label to the corresponding area. While this requirement is not very specific, it is clear that the point must be within the area. We have addressed

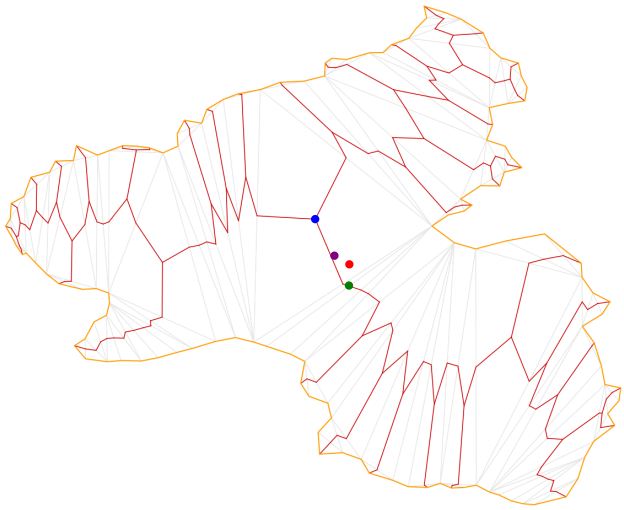

Figure 4.7.: Finding representative points for an area with 4 different methods.(© L. Baur [Bau19])

this issue in the bachelor thesis of Lukas Baur [Bau19]. Figure 4.7 illustrates an example with the computed center points of four different computation methods.

Different standard techniques can be applied, like the geometric center, morphologic erosion and the point of inaccessibility (see [Bau19] for details). Computing the geometric center gives optimal results for convex polygons. But this case is very rare in real world data sets. The point of inaccessibility as well as the morphologic erosion perform good in many cases, but may perform badly in case of long but relatively thin or ramified areas. In these cases, the analysis of a polygon skeleton turns out to perform well. This is a fourth class of proaches, where Baur evaluates several different analysis methods.

Summed up, Baur supposed to use an input sensitive approach. Depending on the input polygon, different algorithms are applied to find candidate positions. Finally the best of these candidates is returned as the best representative point. For details please refer to the original work.

The approach, described above, performs good finding a representative point for an area. But it does not integrate very well into the area labeling scheme. When zooming out a labeled area, the representative point may be very far from the actual placement of the area label. So the user can hardly see any connection between the vanishing area label and the corresponding point label.

A possible solution is to take the lower center of the label box as the representa-

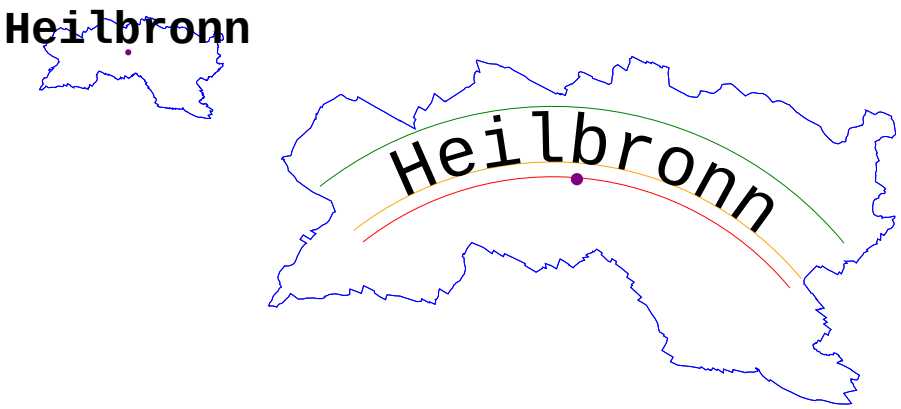

Figure 4.8.: Heilbronn labeled by an area label (right). The representative point is placed at the center of the lower box boundary (purple dot). On the left hand side the area is labeled with a point label on a smaller map scale.

tive point. In this case the point label and the area label integrate well (see Figure 4.8).

5. Labeling Points

While area labels, as introduced in the former chapter, are used to label large entities on a map, there may be objects of very small extent which are of interest for a user as well. On small scaled maps, these may be for example cities. On maps of larger scale, showing e.g. a city area, objects like schools, restaurants, cafés etc. might be of interest for the user. On even larger map scales, ATMs, trashbins, benches and the like could be labeled separately.

Some of these entities may be represented by their name (e.g. cities), while others may have an icon assigned (e.g. trashbins, benches). What they all have in common is that we can assign some sort of importance or priority to the entities. A city is probably more important to be labeled than a village. Also a city with a population of a million might be preferred over one of a hundred thousand.

Figure 5.1.: A very basic point label model where the label is aligned to the point at its lower left corner.

Consider a very basic labeling where for each point label the label string is horizontally aligned and anchored at the point it labels. An example is illustrated in Figure 5.1. The main goal of a good labeling is to prevent these point labels from overlapping each other.

The labeling of points in interactive maps rises two more challenges. The first challenge is related to map zooming. When zooming into the map continuously, the displayed labels grow relative to the map and might start to overlap. In this case, one of the two overlapping labels needs to be removed from the view to regain good readability. An associated problem may arise on further zooming. If the other label gets removed too, the earlier removed one might have enough free space to be displayed again. But displaying this label again would be inconsistent for the user. Hence it must be prevented.

The second challenge is related to map rotation. Consider a map at a fix map scale. When rotating the map, horizontally aligned labels might start overlapping each other. Hiding one of the corresponding labels to resolve this conflict might lead to flickering effects during continuous rotation. These flickering effects might distract the user and hence also need to be prevented.

A preliminary version of this Chapter 5 was published in [Kru18].

Been et al., in their works about labeling points of interest, [BDY06; Bee+10], defined some consistency criteria for interactive map labelings. While they were just considering non-rotating interactive maps, we will use these criteria (he calls them desiderata) as the basis of our proposed labeling scheme. We will shortly introduce them in the following Section 5.1. After that we will introduce our labeling model in Section 5.2. The link between the labeling and map zoom is described in Section 5.3. Section 5.4 finally describes some extensions which can be used to overcome some drawbacks of the labeling model.

5.1. Consistency Requirements

Based on the consistency desiderata defined by Been at al. in [BDY06], we define the following requirements for the labeling of interactive maps which allow panning, rotating and zooming of the map view:

(D1) *During monotonous zooming labels should not appear and disappear more than once.* This requirement meets the user expectation that on continuous zooming, an object is visible until it is no longer important enough (when zooming out). When zooming in, it ensures that a label can vanish, e.g. if the labeled object covers the whole view or the labeled point got replaced by an area or line feature label at larger scales.

(D2) *Labels should not change position or size abruptly on map interaction.* Abrupt label changes during map interaction may distract the user and make it difficult to track the position of the labeled objects.

(D3) *During panning and rotation, labels are not allowed to appear or disappear except for moving in and out of the view area.* This implies that labels might be partly visible if the label is not fully contained in the view area. Especially in navigation systems, the map rotation changes automatically, e.g. when it is linked to a driving direction. This requirement ensures that a constant labeling is visible even if the map rotation changed since the last time the user looked at the map. Hence it allows the user to keep orientation with a low cognitive load.

(D4) *The label placement and selection is a function of scale and the view area. It does not depend on the interaction history.* With requirement, the map at a specific map setting looks like a static map labeling. So users might directly recognize the places if they look at the map, when having recovered their map setting.

Based on the work of Kreveld et al. [KOS97], we add another constraint. This targeting the fact that there is some inherent order of precedence for geographic features to be labeled.

(D5) *During zooming a label disappears only if it is in conflict with an equally or more important label.* For example a megacity label is preferred to a label of a small rural settlement. So if those two labels are in conflict, the megacity label should be kept instead of the settlement label. On larger map scales a street name is less important than the label of the city in which the street is located. During the label selection process, these precedences need to be taken into account. Our label selection process additionally respects what Kreveld et al. called the *relative importance* of an object. For example a town might have a high relative importance if it is in the middle of nowhere compared to a city that is located directly beside a megacity with millions of inhabitants. The relative importance manifests itself in the fact that the label of the town is shown longer than the city label while zooming out of the map.

5.2. The Label Disk Model

We are considering a set of points of interests. For each point, we are considering its label that may be a label string, an icon or the like. We also assume to have a priority function. For two points it is defining if the one is prioritized over the other or vice versa or if they are equally important. The problem we are faced with is the following: given a specific view area, scale and rotation angle, select a subset of points such that a visualization of the corresponding labels fulfills the requirements as defined in Section 5.1.

In the following, we will describe the labeling model at first and a label selection process which allows to efficiently retrieve such a point set. This selection process can be subdivided into two phases: first the label set is preprocessed such that in a second interaction phase the actual label selection reduces to a simple filtering step. This allows to efficiently query the data set for a consistent labeling during the interactive visualization phase.

In order to fulfill the consistency criteria, we define a label disk for each of the points. The label disk is centered at the corresponding point location and has a specific radius depending on its label size, i.e. the label length, the font and font size or the icon and icon size. We require the label to be completely contained within the corresponding label disk. This should hold for each rotation angle.

In order to fulfill requirement **(D2)** and **(D4)**, the label placement must be a function of scale and rotation angle. It must ensure that the label does not

change its position and size abruptly during map interaction. Except for these restrictions, the concrete label placement within the label disk is unconstrained. A fairly simple example for such a placement function, which fits the idea of the model well, is depicted in Figure 5.2. There you see a point label which is horizontally aligned and centered above the labeled feature. During rotation the label remains horizontally aligned and keeps its absolute size on zooming. Icons can be placed centered at their location or centered above the location. Of course many other placements are also possible.

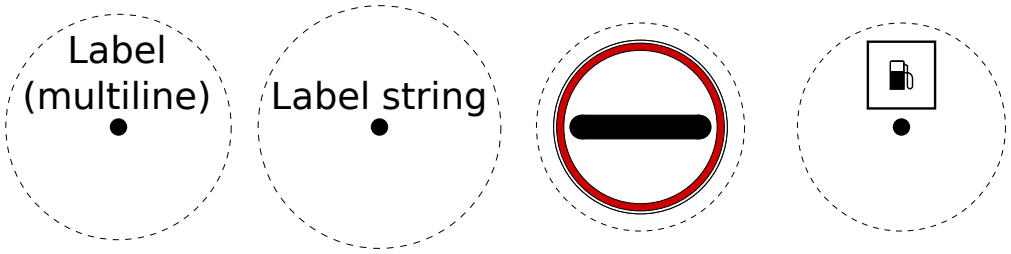

Figure 5.2.: Example of point of interest labels with label strings (left) centered above the labeled feature and icons centered at the labeled feature and centered above the labeled feature (right - [Con17b]). The associated label disks are depicted by the surrounding dashed circle.

Using the label disks, we define a consistent labeling to be a subset of the labels such that the corresponding label disks are non-overlapping. Because each label is completely contained within its corresponding label disk by definition, this ensures that none of the labels are overlapping in any rotation of the map view. Hence we ensured that during rotation none of the labels need to disappear to avoid label overlap. In Figure 5.3 you can see a visualization of Germany labeled with our scheme in two different rotation angles.

To ensure consistency during panning, we come back to a concept Been et al. called an "inverted sequence" in their approach in [BDY06]. The intuitive label selection and placement method is to first select the subset of labels in the view area and placing the corresponding labels afterwards. As Been et al. argued in their paper it is hard to achieve interactive speed and consistency with this approach. We suggest to first pick a consistent labeling globally. From this restricted label set, we finally display the labels intersecting our view area. This selection process ensures that the only way labels appear or disappear on panning is by moving in and out of the view area.

In summary until now our requirement **(D3)** is fulfilled, i.e. labels only appear and disappear by moving in and out of the view area. The requirements **(D2)** and

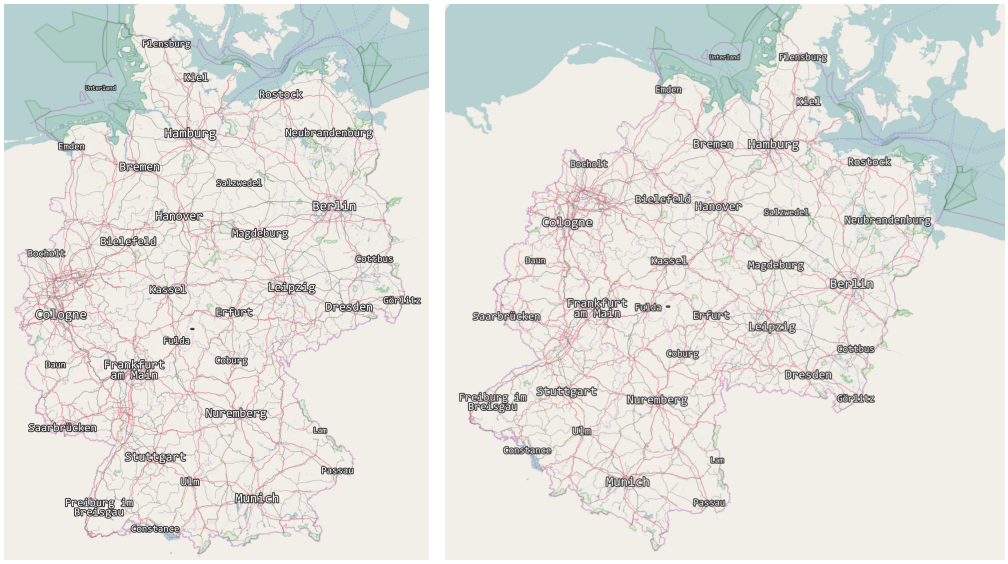

Figure 5.3.: A labeling of Germany in two different orientations (© OpenStreetMap contributors). The basic label set contains all human settlements extracted from the OpenStreetMap dataset [Con17c].

(partially) **(D4)** are fulfilled by an appropriate label placement function. For the latter we did not yet define the dependency to the map scale but we are going to make up for it right now.

The map interaction we haven't considered yet is zooming. Zooming out of the map by decreasing the map scale naturally leads to decreasing the level of detail of the map, i.e. less details are visible and labeled on the map. To support this, we define the label disk radii to be dependent on the map scale. Instead of the label radius being a fix radius r, we define the actual disk radius of a point to be $r \cdot \frac{1}{s}$, where s is the current map scale. You see that decreasing the map scale s enlarges the label disks. Since we require them to be non-overlapping, a consistent labeling contains less labels – the level of detail decreases. By using these scale-dependent label disks, the selection of a consistent labeling gets a function of scale as required in **(D4)**.

The defined label model now allows to have a labeling at arbitrary map scales. But we have not yet taken the actual selection into account. What we also have not yet taken into account are the consistency criteria concerning the zooming process itself. As defined in the consistency requirements **(D1)** and **(D5)**, for the zooming we have some requirements. A label should not appear and disappear more than once during monotonous zooming. Additionally we required a label to be removed from the view only if it is in conflict with a higher or equally prioritized label. We

will target this in the selection procedure described in the following section.

5.3. Linking Labeling and Map Scale

We already defined a labeling model that ensured some of our requirements to be fulfilled. These concerned panning and rotating on an arbitrary map scale. We now want to focus on the process of selecting consistent labelings such that the remaining requirements are fulfilled. These are concerning the process of zooming in and out of the map, respectively increasing or decreasing the map scale. The following two requirements still need to be fulfilled: **(D1)**: 'During monotonous zooming a label should not appear and disappear more than once' and **(D5)**: 'During zooming a label disappears only if it is occluded by an equally or more important point label'

For the sake of simplicity, we only consider the process of zooming out of the map, respectively decreasing the map scale. It is straightforward to think of the following observations in the reversed process of zooming in. Furthermore, we will look at the label disks only and not rely on the actual label placement. We assume that this is done in a suitable way as described in the previous section.

In order to find a proper consistent labeling for a target map scale S, we are considering the following process: starting with a sufficient large map scale s, all the corresponding label disks are free of intersections because all the radii tend to 0. We continuously decrease s, until two of the label disks touch. Now the priority function comes into play. If one of the corresponding points is prioritized over the other, we remove the less prioritized one. Otherwise we remove one of the two. We continue with the process until $s = S$, i.e. our target scale is reached.

The process immediately ensures requirement **(D5)** to be fulfilled, as a label is removed only if its label disk is in conflict with a label disk of an equally or more important label. Requirement **(D1)** is also fulfilled by design of the label selection as a label never reenters the process after being removed once.

The proposed label selection process always leads to the same label "elimination sequence". At least if we assume the label set to be immutable and breaking ties happens deterministically. Each label can be assigned a specific map scale where it is removed from the label set during the process. This opens space for our promised precomputation phase. Because the label elimination sequence and the elimination scales for the labels do not change, we can compute them in advance. Having computed them for a set of labels we can derive a consistent labeling as follows: for a given map scale S, we choose the subset of labels having an associated elimination scale smaller than S and restrict the subset to those labels intersecting the view area. This allows to retrieve a consistent labeling at an interactive speed.

Looking at the process from a more abstract point of view, we are presented with

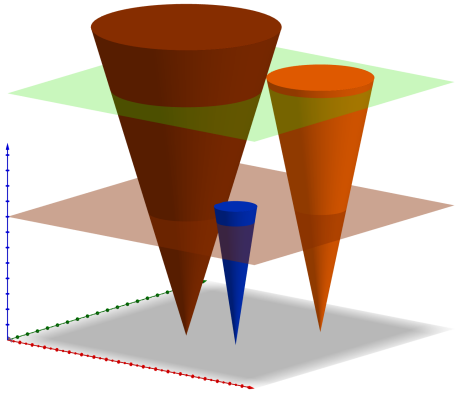

Figure 5.4.: Label cones and two planes (brown, light green) that correspond to label selections at different map scales.

the following so called space-scale cube. The labels are located in a 2-dimensional plane. Using $\frac{1}{s}$ as a third dimension, we see that each label is induces a cone when decreasing s (see Figure 5.4). The elimination scale of a label determines the height of the cone. The cones themselves do not intersect each other. In this view, a labeling of a map on a specific scale S corresponds to the intersection of the label cones with the plane at the height of $\frac{1}{S}$. In the referenced drawing, you see two planes corresponding to two different map scales in brown and bright green. The intersecting cones with the plane are corresponding to the label disks of the labels at the specific map scale.

Since we always keep the whole disk space free of overlappings, the map space is not densly covered. This is not so much a problem in interactive maps because the reduced level of details can be compensated by zooming the map view. In our running example, the navigation context, this may even be an advantage. By not distributing the labels so densely, the map is much easier to read. Nevertheless there are some extensions to the presented labeling scheme, which allow to address this drawback. In the following we will discuss some of them.

5.4. Extending the Model

What we did not take into account yet is the point where a label occurs while zooming out of the map. In the basic labeling model, we described before, all the labels are visible at the largest map scale. A straightforward approach, to extend the labeling model, is to introduce a “popup scale” for a label. It means that the label is not visible from the beginning of the process, but becomes visible at this

specific map scale. At larger map scales, the label does not exist, hence does not occlude any of the existing labels. This concept for example allows to add a label of a city on smaller map scale only such that the label does not occlude details of the city while being in a zoomed in map view. This extension does not violate the requirement **(D1)** as the label only appears once during a monotonous zoom. An example of a labeling of the center of Berlin is depicted in Figure 5.5, where the Berlin label is introduced at the smaller map scale on the right.

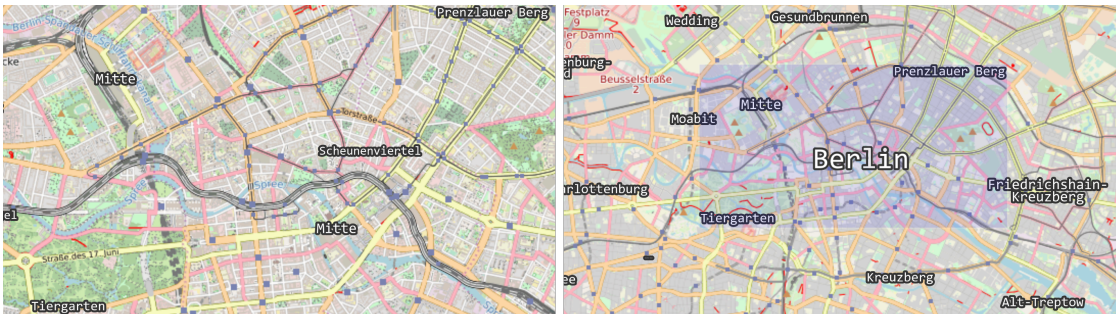

Figure 5.5.: Labeling of a map using a map labeling with popup scales at a larger (left) and smaller map scale (right) where the “Berlin” label popped up (© OpenStreetMap contributors).

As described above, the labeling scheme is not focused on maximizing the number of labels visible, but on visibility and readability of the map with low cognitive load. For example for simple horizontally aligned labels a lot of the available disk space remains unused. To increase the number of labeled objects one can think of a multilayer approach as follows. Each label, which is removed while zooming out, is moved to a second label layer. Labels in this second layer may be overlayed by the main labeling layer. In this second layer labels might use another font type, color or opacity to make clear it is a background layer. Technically the second layer uses the same labeling model, i.e. the disks of labels in the second layer do not overlap. For each label the elimination scale in the first layer is the popup scale in the second layer. So for each of the layers, the consistency requirements are fulfilled but the labels of the different layers might overlap in the visualization. The two layers are visually clearly distinguishable. Hence the concept should not distract the user much. At the same time, it increases the number of displayed labels. In Figure 5.6 you see an example of a labeling with two layers.

Having in mind the concept of popup scales as described above, we introduce a third extension. As Imhof pointed out in [Imh75], line or area feature labels turn into point labels on smaller map scales. For example, a church might be displayed as an area on larger map scales but on a zoomed out map view its area degenerates to a single point. Analogously, the label needs to turn from an area

Figure 5.6.: Labeling of Stuttgart that uses a second label layer (filled gray font) to increase the number of labeled points (© OpenStreetMap contributors).

label to a simple point feature label. In Figure 5.7, you find a map in two different map scales. The area of Berlin is labeled with an area label on the left hand side while it is labeled as a point object on a smaller map scale (right). The same can be applied to line segment labels. By using the concept of popup times, we can represent this effect with our point labeling scheme. Simply setting the popup time for the point label to the map scale where the area label becomes to small to be read, allows us to support this visualization feature.

Figure 5.7.: Labeling of an area using an area label (left) and a point label at smaller map scale (right)(© OpenStreetMap contributors).

6. Obtaining Geographical Data

When we first started working on the labeling scheme, one of the early questions was how to obtain data records of geographical data. Several sources disqualified because they were limited in scope, e.g. did only contain names of human settlements or shops etc. Some were locally limited, i.e. available only for one national state or similar. Others were excluded because they were not freely accessible. We finally came to the OpenStreetMap project [Con17c]. This project maintains a huge set of geographical data which is collected voluntarily and provided as open data. The data is very detailed at least for large parts of the world, see Figure 6.1 for an example of our hometown stuttgart.

A huge advantage is the amount of tools available in the OpenStreetMap ecosystem. Just to mention the most important ones in our case: The *Geofabrik* [Gmb18] provides local extracts of the complete data set which allows to use subsets of nearly arbitrary size. The *osmpbf* project [inp19] provides a parser for the binary format used to distribute the data. Last but not least the *openlayers* project [Con17a], beyond others, provides a framework to visualize geographical data and is free to use.

The data set contains three kinds of objects. *Points* are used to represent objects which are located at a particular geographic position. Examples are trees or other small scale objects like ATMs, trash bins etc. Besides such objects many large scale objects are also modeled as a single points, e.g. cities and other settlements. Additional information, like name (in several languages), point type, population and many more, is attached to the object. Based on these informations its easy to define a corresponding label, the font rendering parameters or an icon and its size.

Ways are representing objects like streets, i.e. a list of geographic positions defining the geometry of the object. The labeling of line-like objects is a separate field in map labeling which we do not consider in the work at hand. Even though it is possible, we did not use point labels to represent such objects on small-scale maps. This is mainly because the scope of application is limited to very specialized scenarios. Additional to representing line-like objects, segments are also used to determine the shape of the third kind of objects: areas.

Areas are representing objects with a 2-dimensional extent. These may be buildings or larger scale objects like city districts, cities, states, nations and many more. Areas are obviously used as input for area labels, if wanted. But areas can also be used to generate point labels for small-scale maps. For example an important

Figure 6.1.: OpenStreetMap view on the inner city of Stuttgart showing the high degree of detail of the data set.

historic site may be labeled with a point label on maps of smaller scale.

In the work at hand, points of interests were created from tagged OpenStreetMap points. The available name tag was used as the label. If the names were long, they were splitted into two lines for minimizing the required disk radius. For settlement points, a priority was derived from the point type. Further information like population size was used to distinguish different priorities for larger and smaller towns and cities. Besides these settlement points of interest, we also exported general point of interests. For example restaurants, shops, bars, ATMs and much more. Since they did not have names assigned, we used icons for representation.

For each of the points, a font size was specified so that the label style correlates to its importance. The specific disk radii where computed by measuring the exact sizes of the label in its corresponding font style. In case of the icons, the disk radii were set according to the icon size.

Areas were considered mainly in case of administrative areas. They were tagged from administrative level 2 (national border) to 11 (neighborhood). Besides these administrative areas, many important sites were modeled as areas as well. We were using the simple geometric center to derive a representative point which we used for a point label. A better solution to do so was presented in Section 4.4.

Part III.

Algorithms

7. Area Label Positioning

As described in the above section 4, the problem of labeling polygonal areas with a curved label can be reduced to finding a box of maximum extent within the polygon. The length to height ratio of the box is hereby determined by the label. The problem is to find a suitable base arc and the best position of the label box along this arc. A label position can be described by a circle center, an inner and outer radius and two angles determining the position of the box along the circle. See Figure 7.2 for an illustation. The label box is drawn in yellow.

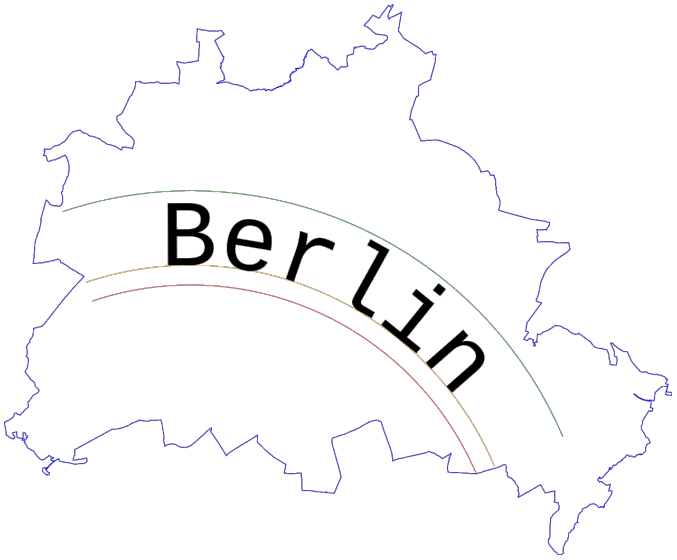

Figure 7.1.: The area of Berlin with an area label and the corresponding box.

In the following section, we will define the problem formally. A quick review of an algorithm proposed by Barrault in [Mat01] is given. His approach is the direct predecessor to the work at hand. We describe some shortcomings of his approach and introduce modifications which we made in order to boost the algorithm efficiency and minimize the computation time. Implementation details and experiments are given at the end of the chapter.

The research described in this Chapter 7 was joint work with Thomas Mendel. A preliminary version of this chapter was published in [KM20].

7.1. Problem Description

We are given a polygon which potentially contains holes. The latter are also given as polygons. We suppose that the polygons do not intersect itself nor any of the other (hole-) polygons.

The following is the input of our problem:

- A closed polygon:
$$(p_1, \ldots, p_n, p_1), p_i \in \mathbb{R}^2$$
- A set of holes (potentially empty) of the same form:
$$\{H_1, \ldots, H_m\} \text{ with } H_i = (h_1, \ldots, h_l, h_1), h_j \in \mathbb{R}^2$$
- A ratio of the area label bounding box:
$$A = \frac{H}{L}$$
where H is the height and L the length of the bounding box of the label.

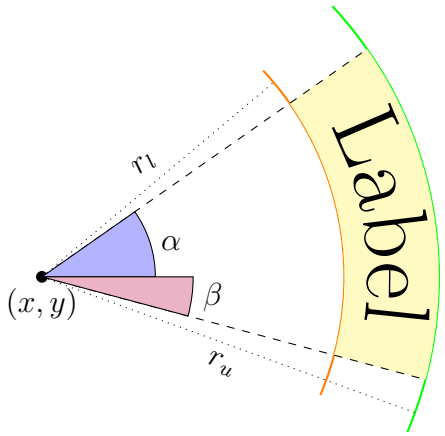

Figure 7.2.: Meaning of the 6-tuple describing a curved area label position (in yellow): $(x, y, r_l, r_u, \alpha, \beta)$

Our goal is to find a placement of a box within the area polygon such that the box does not intersect the boundary polygon nor any of the holes. The box is allowed to be bend along a circular arc but needs to have the given length-to-height aspect ratio. We want to maximize the box size. More formally: we need to find the box position and extent i.e. a 6-tuple:

$$(x, y, r_l, r_u, \alpha, \beta)$$

where (x, y) is the center of the circular arc. The two radii r_l, r_u are the lower and upper radius and α, β is the start and end angle of the box. See Figure 7.2 for an illustration.

The computation of the optimal label placement comprises two major challenges. First, there are infinitely many circular arcs intersecting the polygon. We need to find the most promising ones. Second, a placement of the box along the circular arc needs to be determined which allows to grow the box to a maximum extent. So the main idea of the algorithm is as follows: We first have to find some arcs, which are a reasonable fit to the polygon. To find such an arc, we use an approximation of the medial-axis - a polygon skeleton. A long path through this graph should be an appropriate representative of the area's shape. Because we want our labels to be placed along circular arcs, we fit a circle through the vertices of the path. A set of candidate arcs are evaluated i.e. the optimal label-placement along the arc is computed. The label placement which is largest in size is reported.

The algorithm, Barrault introduced in 2001 in[Mat01], is exactly as stated above. In the following we will review his algorithm in Section 7.2. We further discuss the main drawbacks of his approach which we are going to optimize later on. In Section 7.3, we present our improved algorithm in detail. After that, in Section 7.4, we give some details of the implementation and some experimental results.

7.2. Barrault's Incarnation

Barrault's algorithm follows the main steps described above. To decrease the complexity of the input polygon an additional morphologic erosion is applied. For the eroded polygon, a delaunay triangulation is computed. For each of the delaunay triangles, a convex-combination of its corners defines a "center point" of the triangle. Those "center-points" of adjacent triangles are connected and thus form the edges of the skeleton.

After approximating the medial-axis in this manner, the 50 longest shortest paths are selected. A circle is fitted through each of them and further investigated as a possible label support line. To further reducing the candidate set, the perceived coverage of the 50 candidates is computed (see Section 4.1 for details). The 10 highest ranked candidates are investigated in detail.

For each of the candidate support lines, an optimal label placement and is computed. The placement with the highest score is returned as the optimal label.

To find an optimal placement along a support line, every possible combination of start angle and spacings is considered. Since these are potentially infinitely many, they are investigated in a discretized manner. A potential label baseline is defined by a starting and ending angle defining the start and end of the label on the support line. The two angles are constrained by the start and end of

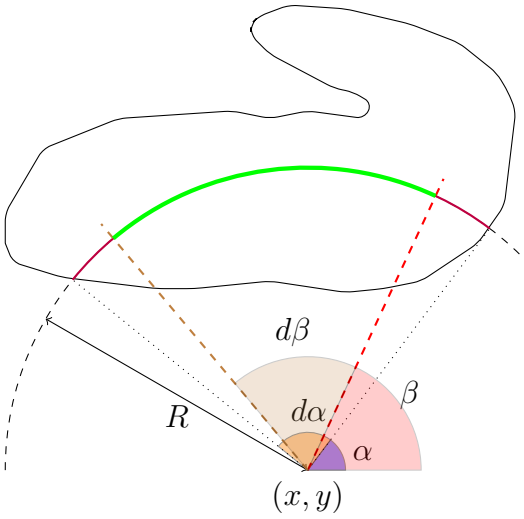

Figure 7.3.: Baseline of a label (green) determined by the support line (purple) and the angles β and $\beta + d\beta$.

the support line (see Figure 7.3). To evaluate an actual label placement (i.e. a concrete baseline), the perceived coverage of the label is computed. This is done by integrating the minimum of the upper and lower distance to the polygon along the label base line in a discretized manner. The contribution is negative for the parts of the baseline ranging into the first and last sixth of the support line. For details please also check Section 4.1.

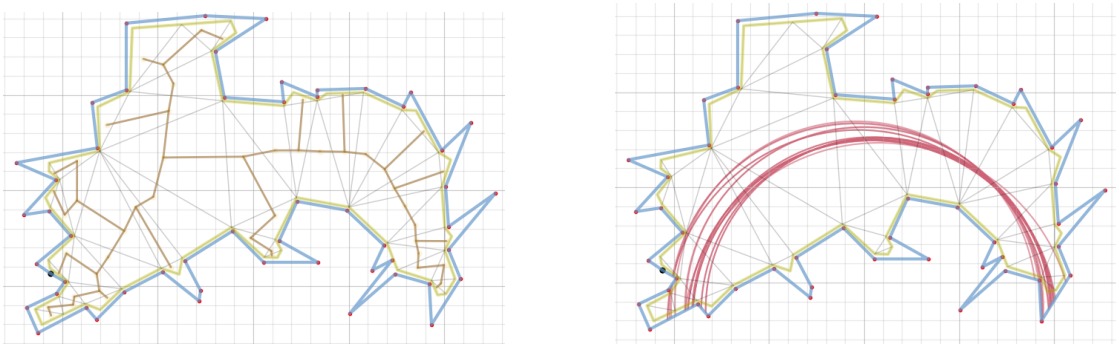

Figure 7.4.: Polygon skeleton (left) computed according to Barrault's approach. The 50 longest shortest paths induce very similar candidate arcs (right). The illustration were used with permission from [Men18].

A major drawback of Barrault's approach is his choice of the 50 paths he is evaluating. These paths are mostly very similar and so are the fitted circular arcs (see Figure 7.4). N. Mendel reproduced this in her bachelor thesis [Men18]. As

a result many promising alternatives are not considered at all. Additionally, the evaluation of the possible label placements contain several integral computations. Each requiring much computation power. Overall the computation takes a long time and in many cases does not even lead to optimal results.

7.3. Real-time Area Label Fitting

We go beyond Barrault's algorithm in several points. Firstly, we use a skeleton based on the Voronoi graph. The reader may want to see 3.2.1 for an explanation. This allows for each edge in the skeleton to compute the minimum distance to the boundary polygon. We call this distance the clearance of an edge.

This clearance value is then used to find paths in the skeleton which are promising to fit a large label through. This discards paths which are too close to the boundary of the area, what would restrict the label size. We also improve path selection by computing a more diverse set of paths.

A third improvement is related to the finding of an optimal position of the label along a candidate arc. Here we are proposing a new scheme to compute an optimal placement along the arc.

7.3.1. Medial Axis Approximation

To get an approximation of the medial-axis where we are able to bound the distance to the boundary polygon we proceed as follows: We compute the delaunay triangulation of the boundary polygon. For each delaunay triangle, the Voronoi center is defined as the center of the circumcircle of the delaunay triangle. We connect the Voronoi centers of adjacent delaunay triangles if this so called Voronoi edge is completely contained within the polygon. For these edges, we can approximate the clearance, i.e. the minimum distance to the boundary polygon. We need to distinguish two cases: If *the Voronoi centers are on different sides of the delaunay edge*, the minimum clearance is half the length of the radical line of the two circumcircles (i.e. the delaunay-edge itself). In the second case, *both Voronoi centers are located on the same side of the delaunay edge*. Then the clearance is the minimum of the radii of the two corresponding circumcircles. See figure 7.5 for an example.

The rational behind this is as follows: All points closer to the segment than the computed distance are also contained in at least one of the Voronoi-balls. Those balls are empty of other points by definition of a valid delaunay triangulation. Therefore there are no points within the cleared segment. The clearance of the segment might still intersect the boundary of the polygon. But this can be remedied if the boundary is sampled sufficiently dense. Furthermore we only use these

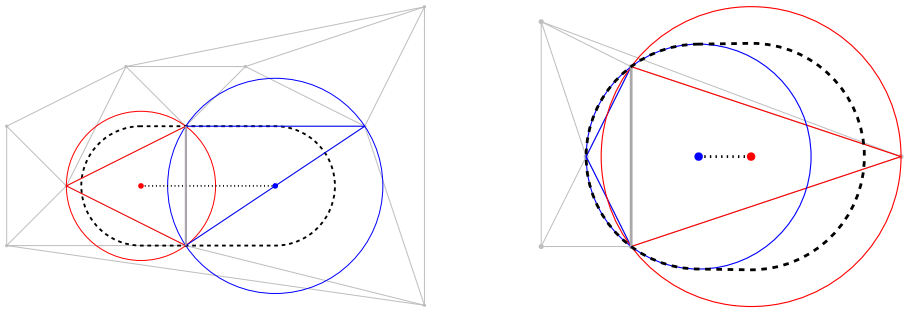

Figure 7.5.: The clearance of a Voronoi edge if the centers are on the same side (left) or on different sides (right) of the delaunay edge.

clearance-values as guidance but do not rely on them for correctness (i.e. keeping the labeling within the polygon).

Having constructed the skeleton graph and the clearance values one is after finding promising paths in the skeleton. This paths should allow to place a label of maximum size.

7.3.2. Finding Candidate Paths

We aim for finding a set of k diverse candidate paths which we further investigate to place a good label. Our strategy is based on the following observation: If we place a label along a given path, the minimum clearance of the path-edges hints at the maximum possible height of the label along this path. We therefore are looking for paths with a high minimal clearance, whose length allows to fully utilize the vertical space promised by this clearance. That is the length of the path should be no less than $l_{min} = \frac{2*clearance}{aspect}$.

The idea is to start with a large clearance value (e.g. the maximum clearance value) and remove all edges of the skeleton which have smaller clearance value. In this subgraph, we search for shortest paths such that their length is larger than the appropriate minimum length. If we can't find enough paths, we reduce the clearance and search for the remaining paths in the subgraph filtered with the new clearance. In our case, we start with the maximum clearance in the graph and reduce it by $\sqrt{2}$, i.e. we half the area of the label box we search for.

In detail we proceed as follows: In each component of the pruned skeleton, we start with an arbitrary node and search for the node which is furthest away. This is done with one dijkstra call by tracking the root node of every shortest-path-tree. The so found nodes form our set of start nodes. We now search for the node which is furthest from our set of start nodes - also requiring only one dijkstra call with all the nodes as sources. The so found pair of nodes approximates the longest

shortest path in the pruned skeleton (this method is exact for trees but not for arbitrary graphs). If the path length is larger than l_{min}, we report the path and add its vertices to the set of start-nodes. If we did not yet find k paths, we repeat the search with the new set of start-nodes. If the found path is of shorter length, we decrease our clearance by $\sqrt{2}$, refilter the graph and proceed as described. We repeat this until we found the k paths.

Through each of the so found candidate paths a circle is fitted. Let $p_1, \dots p_n$ be the points of the path. We compute a center c and a radius r such that the term $\sum_{i=1}^{n} (|p_i - c| - r)^2$ is minimized. We now compute the position and size of the maximum box contained in the boundary polygon and bent along the circle (see the next Section 7.3.3 for details). The maximum-size box is returned as the optimal label placement.

7.3.3. Label Placement

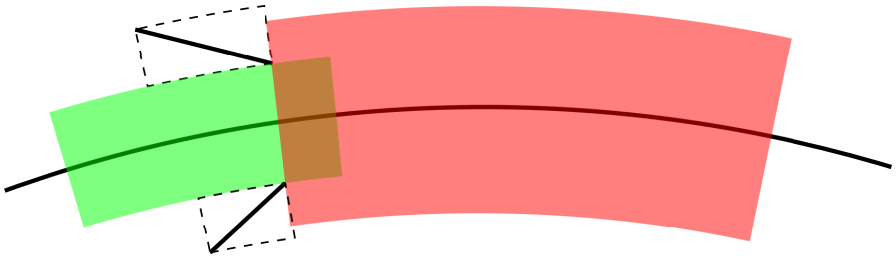

Figure 7.6.: The segments of the polygon restrict the label size. If the label is placed below or above a segment, the segment constrains the possible size (green label). We have to move the label considerably to the side so it can grow (right label).

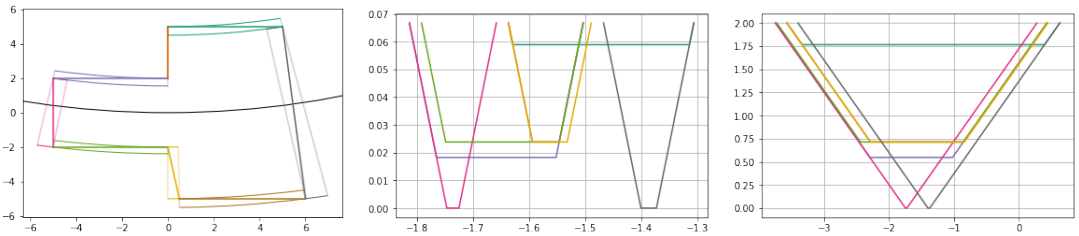

Figure 7.7.: Polygon segments with their circular bounding boxes (left) and the corresponding bounds in the circular diagram for a very tall label (middle) and a very long label (right). The tall label is constrained by the cyan segment. We can actually move it a little to left or to the right. The long label is constrained by the pink and the gray segments.

Given a circle, a polygon and a text label we aim to find the position along the arc such that the size of the label can be maximized. We can compute this optimal placement in time $\mathcal{O}(n \log n)$, where n denotes the size of the polygon.

Let us first consider how a single polygon-segment constraints the label placement. We employ the following simplification: A circular bounding-box is constructed around each polygon-segment.

There are two cases: First if the label size is restricted by the segment in its height, then we can move it along the arc without getting any benefit. In the second case, the size of the label is restricted by the segment in its length. In this case, the size of the label increases if we move the center of the label away from the segment. The more we shift it away from the center, the more we can increase the size of the label. If we consider the possible size of the label as a function of the angle where the label is placed on the circle, we get a piecewise function with 3 parts: When the label gets closer to the segment the possible width decreases until it can fit below/above the segment. It then stays constant, while passing above/below the segment and finally increases. Let's call those functions "wedge". For a given angle, they tell us how large a label can be if it is placed at this angle on the circle. This is illustrated in figure 7.6.

We now construct all those "wedges" from the segments and find the highest point, which is below each of the wedges. This point describes the point where the label width is maximal. Because the label-aspect is fixed, this means that the label size is maximized. So this yields the optimal label placement.

To find this point, we first consider the complete circle from 0 up to $2 \cdot \pi$. We then consider each wedge, from lowest to highest, and restrict the possible placements. When there are no more valid placements left, we return the highest point seen. A simple example instance is depicted in figure 7.7.

The active wedges can be organized in a segment tree. For any height, the set of wedges looks like a set of segments. When going up, these segments grow. When two wedges intersect, we can merge the associated segments.

We can enumerate the wedges with a heap. Because we can stop the computation, when there are no more valid placements left, we only consider a small amount of segments. Considering the example in figure 7.7 with the long label, we would only inspect the pink and gray wedges before returning the optimal placement.

Wedge Computation

In this section, we will go through the math needed to actually compute the wedges. For any mathematical symbols, please consider Figure 7.8 as a reference. Furthermore A denotes the aspect of the label.

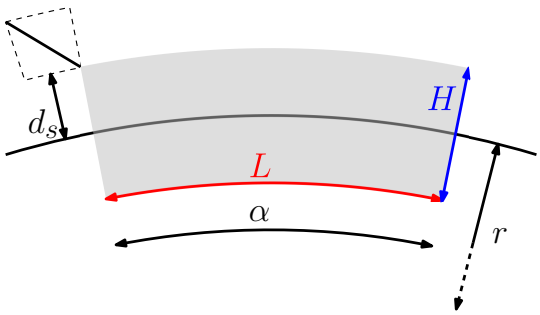

Figure 7.8.: The label can start to grow, when its corner touches the corner of the segment's bounding box.

First let's derive the relationship between the height of the label (H), its width (L) and the spanned angle (α) for a given circle with radius r.

By definition the following holds:

$$H = L \cdot A \tag{7.1}$$

Furthermore we can easily derive:

$$L = (r - H/2) \cdot \alpha \tag{7.2}$$

For a given segment s, let d_s denote the minimal distance of the segment to the circle. If the height H of our label is less than d_s, the segment does not interfere with the label placement. If the height H of the label is greater than d_s, we can compute the spanned angular range α by plugging (7.2) into (7.1) and solving for α. The center of the label needs to be at least $\alpha/2$ from the segment.

With the special case of $H = d_s$, we can compute exactly the placement of the label for which the wedge transitions from one linear function to the next. Coming from the left the label shrinks, until it can fit below the segment. It then slides along without changing size. Finally, its size can increase once again if its far enough to the right.

The following statements are equivalent:

- The height of the label is maximized.
- The length of the label is maximized.
- The area of the label is maximized.
- The angular extent of the label is maximized.

It's easiest to describe the wedges in terms of maximum angular extent.

Let α_{d_s} be the alpha value, such that H equals d_s. Also let β_1 and β_2 be the angles between the circle center and the segment's endpoints. Finally let α_l denote the angle on which the label center is placed.

If $\alpha_l > \beta_2 + \alpha_{d_s}/2$, the maximum possible label extent is:

$$\alpha_{d_s} + 2 \cdot (\alpha_l - (\beta_2 + \alpha_{d_s}/2))$$

If $\alpha_l < \beta_1 - \alpha_{d_s}/2$, the maximum possible label extent is:

$$\alpha_{d_s} + 2 \cdot ((\beta_1 - \alpha_{d_s}/2) - \alpha_l)$$

If α_l falls within those bound, the value is exactly α_{d_s}.

This yields 3 piecewise linear functions for the wedges.

7.4. Implementation and Experimental Results

We implemented our proposed algorithm in `C++`. For the geometric operations, we relied on the CGAL Library [Pro15]. Graph searches were done with the help of the Boost Graph Library [Boo19]. Running times were measured on a single core of an AMD Ryzen 7 2700 and 32GB of RAM.

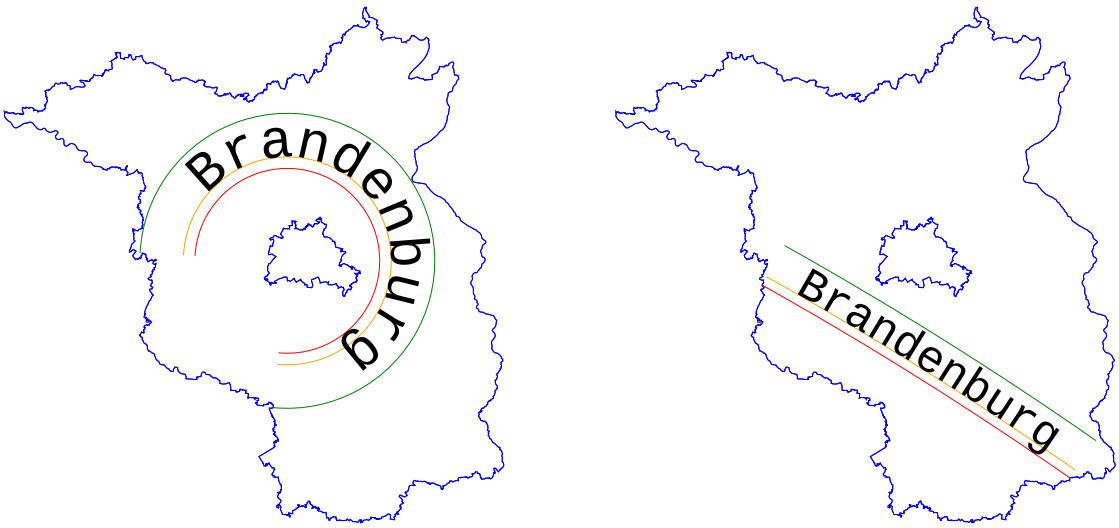

Figure 7.9.: Example labeling of Brandenburg with a label of unrestricted angle (left) and an angle restricted to ≤ 180.

7.4.1. Benchmarks

We compared our algorithm in two variants. In a first run a candidate set of 30 paths was investigated ($S30$). We also ran our algorithm with only 10 candidate paths ($S10$). We also used a Barrault-like search based on the 50 longest paths in the skeleton (BAR). We skipped candidate positions whose arc-length was larger than 180 to prevent labels from being curved to much. In the Barrault-like version, we disabled the skipping since otherwise in many cases no feasible candidate could be found. This can be explained by the low diversity in the candidate set, which Barrault himself noticed in his paper [Mat01] and Mendel reproduced in [Men18].

We evaluated our code on a data set of Germany and its 16 federal states. The data set was created extracted from the OpenStreetMap data. Except for an precomputed data cleaning , no further subsampling or similar was applied. The cleaning was required, since our labeling algorithm requires proper input data without self-intersections and so on.

The number of points at the boundary, and the running time for the area label computation with our algorithm is given in columns *# points* and *S30 runtime* in Table 7.1. For our procedure with a restricted candidate set ($S10$) and the Barrault-like candidate search, the runtime is given. We also state the size of the found box compared to the one computed by $S30$.

When looking at the table, two things attract attention. First, the radio between the results computed from a candidate set of 30 and 10 do not differ. This emphasizes the effectiveness of our proposed candidate selection process. The candidates with high clearing values are contained in both candidate sets. Extending the candidate sets with path candidates of smaller clearing value does not influence the quality of the results. Hence, we can conclude that the optimal position in these cases is indeed induced by one of the paths of high clearing.

Second, the ratio of some results computed using the Barrault-like search is much higher than 100%. But this effect is always caused by the disregarding of the angle restriction. The results for the Brandenburg label which is 188% in case of the Barrault-like search is depicted in Figure 7.9. On the left hand side you see the label of unrestricted angle compared to the restricted label computed by $S30$.

The runtimes of the $S30$ and $S10$ variant are as expected, since only a third of candidates need to be constructed and evaluated, the runtimes of $S10$ are lower. On the other hand the runtime for the Barrault-like variant are significantly higher although the number of candidates is less than twice the ones in $S30$. This can be explained by the time required for constructing the candidate paths. Since initially all leaf-to-leaf paths in the skeleton need to be considered for finding the longest ones, a lot more time is required here.

data set	# points	$S30$ runtime	$S10$ runtime	ratio	BAR runtime	ratio
Germany	141,338	2.03		100%		35%
Baden-Württemberg	34,679	0.40s	0.26s	100%	3.27s	76%
Bavaria	117,003	1.71s	1.01s	100%	90.38s	38%
Berlin	5,890	0.10s	0.04s	100%	0.18s	60%
Brandenburg	19,730	0.41s	0.18s	100%	1.04s	188%
Bremen	2,459	0.06s	0.04s	100%	0.08s	58%
Hamburg	2,987	0.07s	0.05s	100%	0.08s	106%
Hesse	21,797	0.26s	0.17s	100%	1.36s	73%
Mecklenburg-Vorpommern	10,154	0.14s	0.09s	100%	0.35s	60%
Lower Saxony	28,789	0.33s	0.23s	100%	2.22s	89%
North Rhine-Westphalia	39,475	0.44s	0.30s	100%	4.58s	60%
Rhineland-Palatinate	24,393	0.28s	0.19s	100%	1.37s	185%
Saarland	6,246	0.10s	0.07s	100%	0.19s	55%
Saxony	37,967	0.46s	0.27s	100%	4.20s	31%
Saxony-Anhalt	13,749	0.20s	0.12s	100%	0.51s	97%
Schleswig-Holstein	5,632	0.10s	0.03s	100%	0.16s	110%
Thuringia	29,332	0.38s	0.22s	100%	2.45s	137%

Table 7.1.: Running times averaged over the number of nodes.

8. Computing Elimination Sequences

As described in the above Section 5, our labeling can be reduced to computing elimination sequences of induced disks in a 2-dimensional space. These disks are centered at the point of interest and their radius is determined by the label size and font parameters.

In the following, we will consider the abstract problem in a more general form where the disks turn into d-dimensional hyper-spheres. Solving this general problem induces a solution for the application by choosing $d = 2$.

In Section 8.1, we will define the problem, analyse its complexity and introduce a simplified scenario. We provide an efficient algorithm for solving this simplified scenario in Section 8.2. We further prove optimality and analyse the algorithm complexity. The provided algorithm depends on abstract spatial queries. We show how to replace these in specific lower dimensional settings and provide experimental results. Section 8.3 switches back to the unrestricted scenario. Heuristics are provided to compute approximations of the optimal result. Their quality is evaluated and compared to several heuristics and to optimal solutions for very small instances ($\leq 1k$ items).

8.1. Problem Description

Our problem is as follows.

Given

- a set of points
$$P = \{p_1, \ldots, p_n\}, p_i \in \mathbb{R}^d$$
- a priority per point (higher value means the point is more important):
$$prio : P \to \mathbb{N}$$
- a radius function:
$$rad : P \to \mathbb{R}^+$$
- a distance measure, which we will refer to with $|p, q|$:
$$dist : P \times P \to \mathbb{R}^+$$

- a popup time where the corresponding sphere appears for the first time. Before this popup time the sphere is not considered in the process. At popup the sphere has a radius of $rad(p) \cdot t = rad(p) \cdot popup(p)$.

$$popup : P \to \mathbb{R}^+$$

- an maximum elimination time, indicating the time where the sphere is eliminated latest:

$$et_{MAX} : P \to \mathbb{R}^+$$

Some information can be derived from the data:

- The collision time for two points $p \neq q \in P$:

$$t_{coll}(p, q) = \frac{|p, q|}{rad(p) + rad(q)}$$

- The maximum collision time t_{max}

Each point $p \in P$ induces a d-dimensional sphere $B(p, rad(p) \cdot t)$ centered at p. The sphere has a variable radius depending on a parameter $t \in \mathbb{R}^+$ and the specific radius of the point $rad(p)$.

We are considering the following process: Let $t = 0$ be the initial t-value. None of the induced spheres overlap (assuming general position) because the spheres all have a radius of 0. We continuously increase t.

During this process three things can happen: The induced spheres of two already popped up points touch at the current t - we say they collide. Let p_1, p_2 be the two related points, w.l.o.g. let $prio(p_1) \leq prio(p_2)$ (otherwise swap the points). If $prio(p_1) < prio(p_2)$, we are eliminating the less important point p_1 and continue. If $prio(p_1) = prio(p_2)$ then either one or the other is eliminated.

Second, the popup time of a sphere might be reached. In this case the corresponding sphere is added to the arrangement. Since the sphere has not been considered before, it might directly intersect many of the other spheres available. Which means the sphere immediately eliminates all spheres with lower priority it overlaps. If it is intersecting with a sphere of higher priority, it is eliminated before affecting any other point.

Last, a sphere could reach its maximum elimination time. In this case the sphere is eliminated and removed from the arrangement.

This process determines an elimination time for each point. We call this function

$$et : P \to \mathbb{R}^+$$

The induced sequence of eliminations is called **elimination sequence**.

This sequence is not necessarily unique. Whenever two spheres collide and the two corresponding points are of the same priority, the point to be eliminated can freely be chosen. Depending on this decision, others might be eliminated sooner or later. Our goal is to maximize the number of spheres that are alive summed over all possible times t. This implies to maximize the integral over t of the number of points alive at time t. But instead of integrating over t, we can sum up for each point the size of the range, where the point is alive, i.e. $\sum_{p \in P}(et(p) - popup(p))$. Since the popup times are constant, this function is maximized if the sum of elimination times is maximized. So our optimization goal is to:

$$\max \sum_{p \in P} et(p)$$

8.1.1. Computational Complexity

We will now prove NP-hardness of our problem at hand. Thus showing that we can not expect to find an efficient algorithm to compute an optimum solution. Efficient in our case means polynomial time for a deterministic algorithm. We use a well known NP-complete problem, the maximum independent set problem in unit-disk graphs, and reduce it to a Elimination sequence computation.

Theorem 1. *Computing elimination sequences is NP-hard.*

Proof. To prove NP-hardness, we choose the Maximum Independent Set (MIS) problem in unit disk graphs. The MIS problem in unit disk graphs is NP-complete [CCJ90]. It is defined as follows: For a given set of unit disks (i.e. disks of radius 1) in the plane, we consider the intersection graph which is the graph where each disk is represented by a vertex. If two circles intersect, the corresponding vertices are connected via an edge. See Figure 8.1 for an example. The MIS is to determine a subset of vertices such that no two vertices are connected via an edge and the result set is of maximum size.

For a given unit disk graph instance, we construct an Elimination Sequence instance as follows: For each disk, we create a point in P. Each point gets a priority, a radius and maximum elimination time et_{MAX} of 1. By checking every pair of points $p \neq q \in P$ with $|pq| < 2$, we can determine the largest distance d between two of these points in $\mathcal{O}(n^2)$. We choose an ε such that $\frac{d}{2} < \varepsilon < 1$. To each point we assign a popup time to ε.

Let us consider the elimination sequence of the constructed instance. Most of the points will immediately be eliminated when popping up at time ε. So $et(p) - popup(p)$ is 0 for these points. Only some will reach their maximum elimination time 1. For these points p it holds that $et(p) - popup(p) = 1 - \varepsilon > 0$. The sum of all elimination times is maximized if and only if the points of a maximum independent

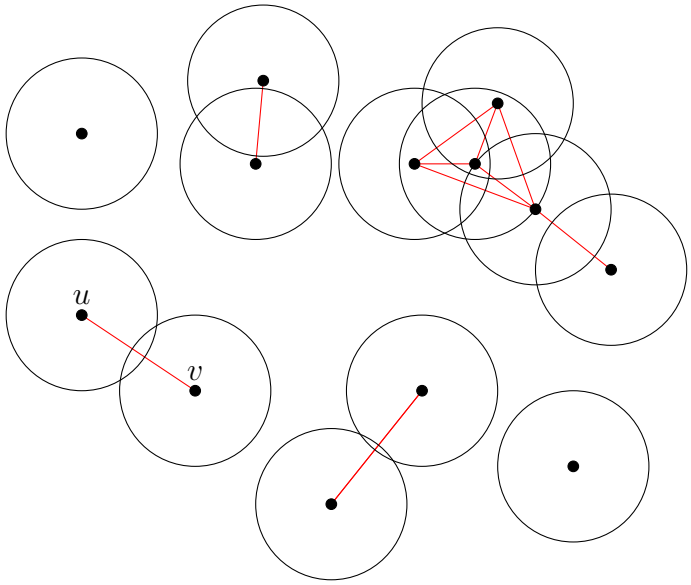

Figure 8.1.: Example of an intersection graph (red edges) of a unit disk set. For each pair of vertices u, v an edge is added, if the corresponding unit disks intersect.

set are not eliminated at popup time ε. Hence an optimum elimination sequence induces a maximum independent set in the unit disk graph instance. The optimum solution to the MIS problem contains the points with an elimination time of 1.

We can conclude that the computation of elimination sequences in 2 dimensions is NP-hard. Since the 2-dimensional case is a special case of the general d-dimensional problem, the theorem follows. □

8.1.2. A Simplified Scenario

The complexity of the problem is due to the necessity to choose which point to eliminate, whenever two points are of equal priority collide. To favor one over the other or vice versa might change the whole sequence for larger t values. The same holds for collisions happening at the same point in time. This might be induced by spheres colliding at the same time or points popping up. The latter may immediately lead to multiple collisions happening at the same time.

By restricting the problem, we can reduce the complexity and find a very efficient algorithm to compute this sequence. In the *simplified scenario* we assume the following:

No two collisions happen at the same time. We can ensure this by slightly perturbing the input if necessary.

The popup time for each point is t=0 , i.e. no point is popping up during the process.

We define the priority function to be a total order by introducing the following additional constraint:

$$\forall p \neq q \in P : prio(p) \neq prio(q)$$

A very naive algorithm can be used to solve this simplified scenario in $\mathcal{O}(n^2 \log n)$ time. The algorithm performs as follows: compute the collision time for each pair of points. Sort the list of elimination events according to the collision time. Start with the event of smallest collision time and handle it, i.e. output the elimination time for the less important point and the point itself. Continue with the next event in the queue: Check if none of both points has been eliminated so far. If so, output the less important one and continue. The dominating operation is the sorting of $\mathcal{O}(n^2)$ elements taking $\mathcal{O}(n^2 \log n)$ time.

After having introduced the problem and the simplified variant, we will now introduce an efficient algorithm in the next section. We prove correctness and analyse the algorithm complexity. The algorithm is using some abstract data structures for spatial queries. After introducing and analysing the general algorithm, we make things concrete for two realistic scenarios: The first is the labeling of maps in a 2-dimensional euclidean setting, e.g. the labeling of a map using the well known Mercator projection. The second scenario covers d-dimensional euclidean spaces. Experimental results are provided for a slightly modified scenario, where the points are located on a virtual globe.

8.2. Solving the Simplified Scenario efficiently

Given the simplified scenario defined above, we aim for computing the correct elimination sequence efficiently. We have already seen a naive algorithm, running in $\mathcal{O}(n^2 \log n)$. But we aim for finding a better algorithm. In the following, we will first gain some knowledge about the elimination process. We will use this in the design of our algorithm. For the sake of simplicity, we will illustrate our examples in the 2-dimensional scenario. These examples easily generalize to higher dimensions.

Our algorithm will mimic the process of the spheres growing from $t = 0$ where the spheres all have a radius of 0 and are therefore non-overlapping (assuming general position). The high level idea is to "predict" the upcoming event for

A preliminary version of the work presented in this Section 8.2 was published in [Bah+17; FKS16].

each point. Processing the smallest of these predicted collisions and update the predictions afterwards, will be sufficient for computing the sequence.

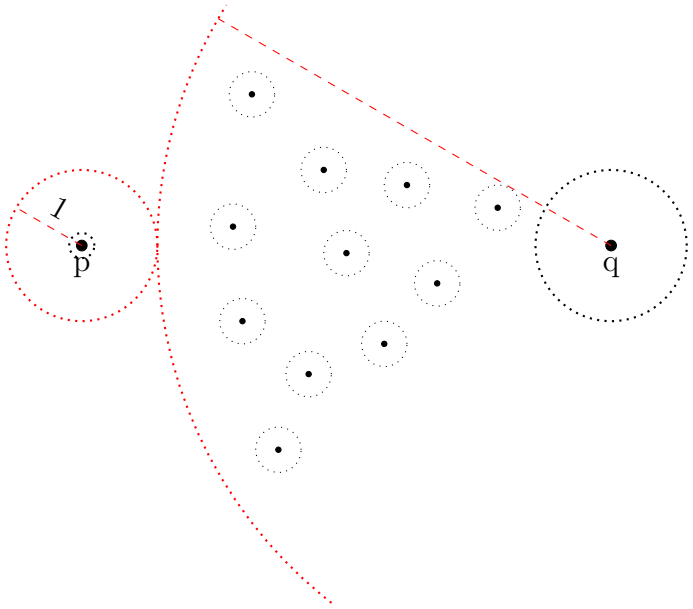

Figure 8.2.: A worst case instance. At an early time (black spheres) it seems like p might collide with a very nearby sphere first. But when the collision finally happens, it is with the most distant point q, which has a very large radius compared to the others.

Unfortunately, the next collision for a point p might be with its most distant point q in the point set. This might be the case if this point has a very large radius $rad(q)$ compared to all the other points in the point set. See Figure 8.2 for an illustration. At the same time, we notice that it is not necessary to predict the next collision for both points correctly. As long as one of the two points correctly predicts its next collision, we are fine. This leads us to the first observation, which will guide our algorithm:

Observation 1. *During the process, only for one of two points of a collision the collision needs to be predicted correctly.*

We will use this to guarantee correct prediction only for the point with larger or equal radius. There is no harm though, if both predict the respective collision.

In the following, let us consider only collisions where p is the point with larger or equal radius. We see that the sphere around p hits the one of its nearest neighbor at half the nearest neighbor distance, assuming equal radii (see Figure 8.3). No other sphere induced by a point with smaller radius can hit the sphere of p earlier. Hence we can conclude:

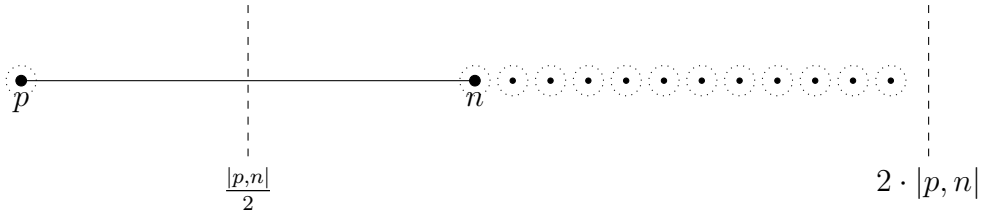

Figure 8.3.: Assuming p to be the point with larger or equal radius, then no other point can collide with p before its sphere covers half the distance to its nearest neighbor n.

Observation 2. *For a point p, the next collision with a point q with $rad(q) \leq rad(p)$ is at time $t_{coll} \geq \frac{|p,nn|}{2 \cdot rad(p)}$, where nn is the nearest neighbor of p.*

We can also consider the other edge case: Suppose $rad(nn) = 0$ for the nearest neighbor nn of p. Then the sphere induced by p collides with the one of its nearest neighbor when it coveres the whole nearest neighbor distance. Every other point inducing an earlier collision, but having a smaller or equal radius than p, must have a distance $\leq 2 \cdot |p, nn|$ to p (see Figure 8.3). Hence we observe:

Observation 3. *For a given point p and its next collision partner q with $rad(p) \geq rad(q)$, it holds that $|p, q| \leq 2 \cdot |p, nn|$, where nn is the nearest neighbor of p.*

These observations at hand we can now head towards describing the idea of our algorithm.

The high level idea for computing the elimination sequence is as follows: a queue manages the upcoming collision events. For each point, the next event in time is predicted and added to the queue. We process the events with ascending collision time. When processing an event, we check if the two involved points are still alive. If so, we output the less prioritized point and remove it from the point set. We then predict and add the next collision for the remaining point.

When predicting the next collision for a point, we can use the two observations 2 and 3. According to Observation 3, we only need to check the range of twice the nearest neighbor distance to find the next collision. This allows us to minimize the number of points, we actually need to check. From Observation 2, we know that we can postpone the prediction until the sphere induced by p covers half the nearest neighbor distance. This might save us some effort as p might have been eliminated in the meantime. Otherwise the number of points we need to check for conflicts might have been decreased - hence decreasing the efford to predict the next collision.

Let us now continue and specify the algorithm.

8.2.1. The Algorithm

For sake of simplicity, we suppose to have given an abstract data structure to answer the following spatial queries. See Section 3.2.3 and 3.3.2 for an detailed introduction.

Definition 5 (NearestNeighbor(q)). *For a point set $P \subset \mathbb{R}^d$ and a query point q, the nearest neighbor of q is a point $p \in P \setminus \{q\}$ with $|p,q| \leq |p',q|$, for all $p' \in P \setminus \{q\}$.*

Definition 6 (Range(q,r)). *For a point set $P \subset \mathbb{R}^d$, a query point q, and a distance r, the range of distance r around q is the set $S = \{p \in P \setminus \{q\} : |q,p| \leq r\}$.*

Definition 7 (Delete(q)). *Removes point q from the point set P.*

In the following, we will state the algorithm, prove its correctness and bound its complexity depending on the complexity of these operations. After that, we will show how to replace this black box data structure in real problem instances.

Our algorithm maintains a min-queue Q which maintains (according to time t) the following types of events in an ascending order:

UpdateEvent $= (p,p,t)$, i.e. check for possible collisions of point p at time t

CollisionEvent $= (p,q,t)$, i.e. a collision between the induced spheres of p and q will happen at time t

Within Q, we resolve ambiguities, i.e. events with identical times t, by considering an UpdateEvent less than an CollisionEvent. Remember: no two collisions will happen at the same point in time. We also maintain an array, called *alive*, to indicate if a point is still alive or has already been eliminated.

In Algorithm 1, we have depicted the main loop of our algorithm. In a first step, the event queue is initialized by calls to PredictCollision$(p,0)$ for each p in the point set. This effectively determines the nearest neighbor nn for every p and inserts an UpdateEvent at time $|p,nn|/(2 \cdot rad(p))$. The time at which we want to check for the next collision of p.

In the main loop, the algorithm always processes the next event from the priority queue. If the event is a CollisionEvent and if both points p and q are alive, then q is eliminated. Which means, its elimination time is set to t, the point is removed from the set of points and $alive[p]$ is set to $false$.

The subsequent if-else constructions perform as follows: If the event was an UpdateEvent(p,p,t), the if clause is processed, i.e. PredictCollision(p,t) is called. Each CollisionEvent(p,q,t) behaves like an UpdateEvent for the not-eliminated point. In case that an elimination took place, PredictCollision(q,t)

Algorithm 1: Main algorithm

Input : $\mathsf{P} = \{p_1, \ldots, p_n\}$,
$rad : \mathsf{P} \to \mathbb{R}^+$,
$prio : \mathsf{P} \to \mathbb{N}$ with $prio(p) \neq prio(q)$ if $p \neq q$
Output: Sequence of tuples $(p_i, t_i) \in P \times \mathbb{R}$

```
foreach p ∈ P do
    Q.push(PredictCollision (p, 0));
    alive[p] ← true;
while !Q.empty() do
    (p, q, t) ← Q.popMin();
    if p ≠ q and alive[p] and alive[q] then
        r ← if prio(p) > prio(q) then q else p;
        output(r, t);
        alive[r] ← false;
        P.Delete (r);
    if alive[p] then
        // an UpdateEvent will be handeled here if alive[p]
        Q.push(PredictCollision (p, t));
    else if alive[q] then
        Q.push(PredictCollision (q, t));
```

is called for the surviving point q. If either p or q were eliminated before, PredictCollision is called for the respective other one. If none of the two are alive anymore the event is just popped from the queue.

Thus, during the course of the main loop, for every point p there is either an UpdateEvent or a CollisionEvent in the priority queue Q.

Algorithm 2: PredictCollision

Data: $\mathsf{P} = \{p_1, \ldots, p_n\}$,
$rad : \mathsf{P} \to \mathbb{R}^+$,
$prio : \mathsf{P} \to \mathbb{N}$ with $prio(p) \neq prio(q)$ if $p \neq q$
Input : Point p, Time t
Output: CollisionEvent (p, q, t) or UpdateEvent (p, p, t)

$nn \leftarrow$ `P.NearestNeighbor (`p`);`
$t' \leftarrow \frac{|p,nn|}{2 \cdot rad(p)}$;
if $t < t'$ **then**
  return (p, p, t') ; `// UpdateEvent`
else
  $nh \leftarrow$ `P.Range (`p, $2|p, nn|$`);`
  $q \leftarrow \underset{p \in nh}{\operatorname{argmin}}(t_{coll}(p, q))$; `// neighbor with minimum` t_{coll}
  return $(p, q, t_{coll}(p, q))$; `// CollisionEvent`

Predicting the next collision of a point p and at time t is done with the PredictCollision routine, stated in 2. First the nearest neighbor nn of p is determined. We compute t', the time at which the sphere around p covers half the nearest neighbor distance. If the current t is less than this t', we return an UpdateEvent to trigger another PredictCollision call if t' is reached. Otherwise, we search the range of twice the nearest neighbor distance to find the next collision in time. We return the corresponding CollisionEvent.

8.2.2. Correctness

We will now prove that the algorithm is correct.

Theorem 2. *Algorithm 1 returns the correct elimination sequence.*

Proof. We show that the following loop invariant holds after each iteration of the main loop: For every future collision p, q, t in the correct elimination sequence, w.l.o.g with $rad(p) > rad(q)$, there is either an UpdateEvent (p, p, t') or a CollisionEvent $(p, ., t')$ in Q with $t' \leq t$, and both p and q are alive.

This suffices for correctness, as at time t the only possible collision is between p and q. Otherwise, the instance would be degenerated. If this CollisionEvent is not in Q, there has to be the UpdateEvent (p, p, t) in Q which leads to the insertion of the respective CollisionEvent upon a call to PredictCollision(p, t). Also, wrong collisions are impossible, as then at least one point needed in a later collision would not be alive anymore.

After the initial loop, the invariant is true as all points are alive and for each correct future collision p, q, t with $rad(p) \geq rad(q)$, there is an UpdateEvent (p, t') in Q with $t' \leq t$. Now assume an UpdateEvent is popped from Q, and $rad(p) \geq rad(q)$ for some future collision of p, q at time t_{coll}. If PredictCollision(p, t) inserts a new UpdateEvent (p, p, t') in Q, obviously $t' < t_{coll}$ holds. Otherwise, a CollisionEvent $(p, ., t')$ is inserted. If it does not predict a collision with q, there has to be some earlier possible collision at the moment, hence also $t' \leq t_{coll}$ holds.

If the first CollisionEvent is popped from Q, it has to be a correct collision as popping UpdateEvents before did not hurt the loop invariant as shown above. Hence the point discarded will not appear in any correct future collisions. For the surviving point PredictCollision is called, which can not hurt the loop invariant as shown above. For any subsequent CollisionEvent, if one of the points is not alive, the collision event reduces to an UpdateEvent for the other point. Hence again the loop invariant is never violated.

□

8.2.3. Algorithm Complexity

In the following, we will analyse the complexity of the provided algorithm. We will hereby rely on the abstract proximity queries. We denote T_{NN}, $T_{RQ}(k)$ and T_{DEL} to be the time required for answering a NearestNeighbor, Range and Delete query in a data set of size n, where k is the size of the requested range. We will later show how these black boxes can be replaced in practice in a two and a multi dimensional scenario (see Section 8.2.4). In the following Section, let the ratio $\Delta = \frac{r_{max}}{r_{min}}$ where r_{max} is the maximum and r_{min} the minimum sphere radius in a given instance.

Proving the running time of the algorithm consists of essentially two steps:

1. Bound the size of the result of the Range query in the PredictCollision subroutine.

2. Bound the overall number of times PredictCollision is invoked.

Let's first bound the number of points having a given point in the point set as their nearest neighbor. We will use this result afterwards.

Lemma 1. *For a set of points P in $\mathbb{R}^d$ and a fixed $q \in P$, the number of points in P which have q as nearest neighbor amongst the points in P is $\mathcal{O}(2^{2.5d})$.*

Proof. For the points p, having q as their nearest neighbor, we know that the circle of radius $|pq|$ around p does not contain any other point. We can map the points p to p' which are on the surface of a sphere with the smallest nearest neighbor distance (see Figure 8.4 for an example in 2-d). For these mapped points the nearest neighbor relation did not change. The spheres around these mapped points of half the nearest neighbor distance are non-intersecting and touch the sphere centered at q with the same radius. The maximum number of spheres in this setting can be upper bounded as follows:

The kissing number τ_d is the maximum number of non-overlapping unit spheres in $\mathbb{R}^d$ that can be arranged so that they all touch a central unit sphere. It is known due to Kabatiansky/Levenshtein [KL78] and Wyner [Wyn65] that $2^{0.2075 \cdot d(1+o(1))} \leq \tau_d \leq 2^{2.041 \cdot d(1+o(1))}$. The latter inequality yields the statement of our lemma. □

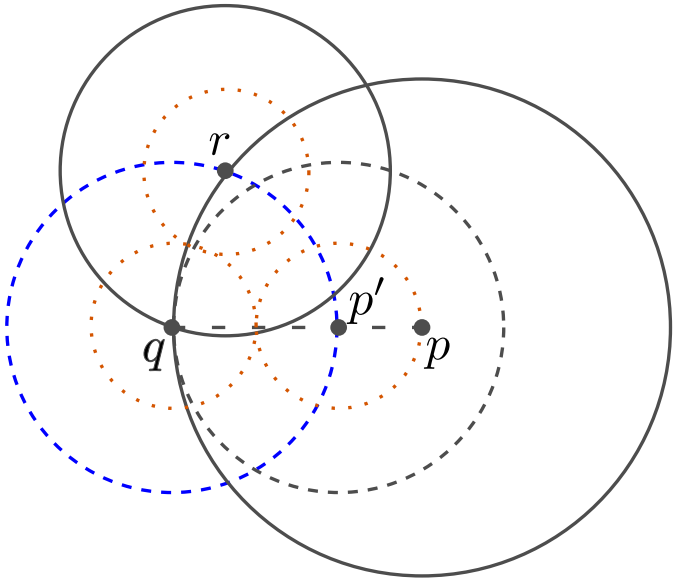

Figure 8.4.: The points, having q as their nearest neighbor, can all be projected on a circle of the smallest nearest neighbor distance (dashed, blue). Their circumcirles of the nearest neighbor distance, do not contain any of the other points. Hence the circles of half the distance (dotted, orange) are non-intersecting and touch the circle of point q.

The following lemma, for arbitrary factors $\alpha \geq 2$, upper bounds the number of points in a range query of α times the current sphere size.

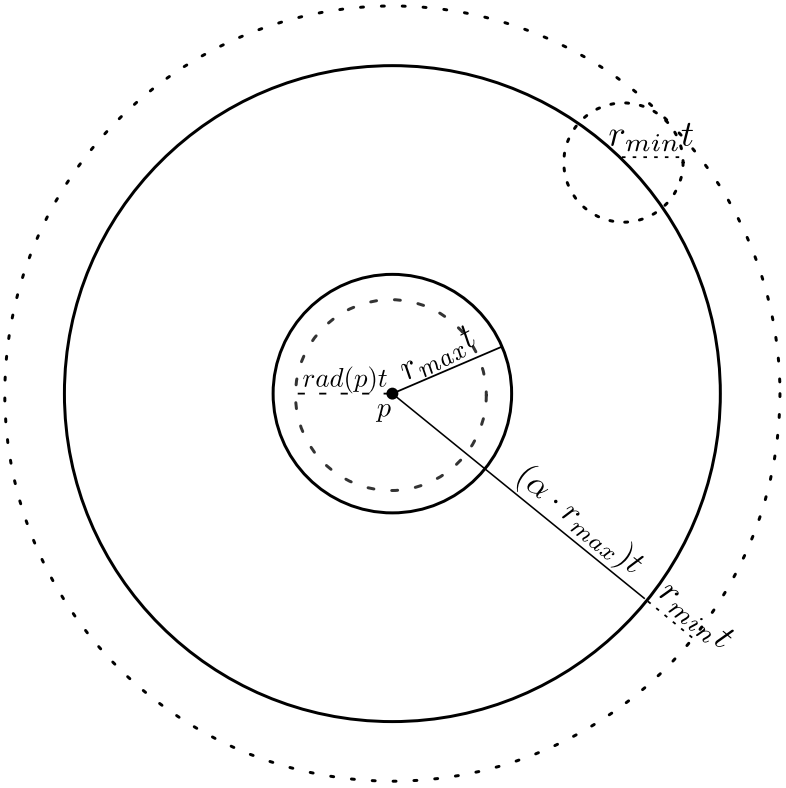

Figure 8.5.: The spheres of radius $r_{min} \cdot t$ around points in $\textsc{Range}(p, \alpha \cdot rad(p)t)$ are completely contained within a sphere of radius $(\alpha \cdot r_{max} + r_{min})t$ - a $2D$ example.

Lemma 2. *In the* PredictCollision *routine for any point p and an arbitrary time $t \geq \frac{|p,nn|}{2 \cdot rad(p)}$ and a constant $\alpha \geq 2$, we have that $|\textsc{Range}(p, \alpha \cdot rad(p)t)| = \mathcal{O}(\Delta^d)$.*

Proof. First consider the volume of a d-dimensional sphere with radius $rad(p)t$ in $\mathbb{R}^d$. In the following let

$$\mathcal{F} = \frac{\pi^{d/2}}{\Gamma\left(\frac{n}{2} + 1\right)}$$

where Γ is Eulers gamma function. Then the volume of a d-dimensional sphere of radius R is $\mathcal{F} \cdot R^d$ in an euclidean space.[Aut, Eq. 5.19.4] For each point in the $\alpha \cdot rad(p)t$-range around p, we know that their spheres do not intersect. It follows that the spheres with radius $r_{min}t$ around the points in $\textsc{Range}(p, \alpha \cdot rad(p)t)$ have to be disjoint and completely contained in a sphere of radius $(\alpha \cdot rad(p) + r_{min})t$. The additional r_{min} is necessary so that the spheres centered at the boundary of the range are still completely contained in the volume. In Figure 8.5, the reader

finds depicted an example in 2d.

Hence the number of points in the range can be upper bound as follows:

$$
\begin{aligned}
& |\textsc{Range}(p, \alpha \cdot rad(p)t)| \cdot \mathcal{F}(r_{min}t)^d \leq \mathcal{F}((\alpha \cdot rad(p) + r_{min})t)^d \\
\Leftrightarrow \quad & |\textsc{Range}(p, \alpha \cdot rad(p)t)| \leq \frac{((\alpha \cdot rad(p) + r_{min})t)^d}{(r_{min}t)^d} \\
& \leq \frac{((\alpha \cdot \Delta r_{min} + r_{min})t)^d}{(r_{min}t)^d} \\
& = (\alpha\Delta + 1)^d
\end{aligned}
$$

The last inequality holds because $rad(p) \leq r_{max} = \Delta r_{min}$. For α being a constant, the result is in $\mathcal{O}(\Delta^d)$ □

With this at hand, we can bound the running time of a single PredictCollision call.

Lemma 3. *The running time of a single call to* PredictCollision(p, t) *is* $\mathcal{O}(T_{NN} + T_{RQ}(n, \Delta^d) + \Delta^d)$.

Proof. In case $t < t'$, the running time is just $\mathcal{O}(T_{NN})$ for the nearest neighbor query.

Otherwise, we additionall inspect the $2 \cdot |p, nn|$-range. In the worst case, this is 4 times the current size of the sphere around p. According to Lemma 2 with $\alpha = 4$, we have to report and inspect $\mathcal{O}(\Delta^d)$ points. This can be done in expected $\mathcal{O}(T_{RQ}(n, \Delta^d))$ time. The bound follows. □

It remains to bound the number of calls to the PredictCollision routine.

Lemma 4. *The number of calls to* PredictCollision *is* $\mathcal{O}((\Delta^d + 2^{2.5d}) \cdot n)$.

Proof. To prove the lemma we distinguish three cases, summing up to the total number of calls:

1. *Calls after an elimination took place.*
2. *Calls from a* CollisionEvent *where the partner is no longer alive.*
3. *Calls from an* UpdateEvent.

To bound the number of calls in the first case is simple: In total $n-1$ eliminations take place, hence $\mathcal{O}(n)$ calls are done from this context.

Let's bound the second case, i.e. number of calls to PredictCollision by a collision event where one of the points has already been eliminated. Consider the

point in time when a point gets eliminated. Every CollisionEvent affected by this elimination is with a point of a distance less than $2rad(p) \cdot t$. We can bound these number of point by $\mathcal{O}(\Delta^d)$, according lemma 2 with $\alpha = 2$. Because this can happen at most $n-1$ times, we get $\mathcal{O}(\Delta^d \cdot n)$ many of these calls.

Finally consider the number of calls to PredictCollision from an UpdateEvent. At the beginning, there are n many such events. If the call to PredictCollision leads to a new UpdateEvent, then the nearest neighbor must have changed since the last call. This can only happen when the former nearest neighbor was eliminated. But a point is a nearest neighbor to a small number of other points as shown in lemma 1. Therefore each elimination induces at maximum $\mathcal{O}(2^{2.5d})$ many UpdateEvents. So overall $\mathcal{O}(2^{2.5d}n)$ such calls to PredictCollision might occur.

The lemma follows. □

So we are able to finally bound the running time of the algorithm.

Theorem 3. *Algorithm 1 has running time* $\mathcal{O}(\xi n(T_{NN}+T_{RQ}(n,\Delta^d)+\Delta^d+\log n+T_{DEL}))$ *with* $\xi = (\Delta^d + 2^{2.5d})$.

Proof. First note that Q never contains more than one event per point, hence all insertion and removal operations on Q have running time $\mathcal{O}(\log n)$.

The initialization requires n nearest neighbor queries and n insertions into Q, hence $\mathcal{O}(n(\log n + T_{NN})$ in total.

The running time of the main loop is determined by:

- the number of collisions multiplied by the time to handle a collision
- the number of calls to PredictCollision times the effort for such a call (lemma 3 and 4) and the time to insert the resulting event.

The number of collisions is $n-1$ and each collision requires the deletion of a point, hence $\mathcal{O}(n \cdot T_{DEL})$ in total.

PredictCollision is called $\mathcal{O}(\xi \cdot n)$ times and requires $\mathcal{O}(T_{NN}+T_{RQ}(n,\Delta^d)+\Delta^d)$ time. Another $\mathcal{O}(\log n)$ time is required to insert the resulting event into the queue. Hence $\mathcal{O}(\xi n(T_{NN} + T_{RQ}(n, Delta^d) + \Delta^d + \log n))$ time in total.

Summing up we have

$$\begin{aligned} &\mathcal{O}(n \cdot T_{DEL} + \xi n(T_{NN} + T_{RQ}(\Delta^d) + \Delta^d + \log n)) \\ &\in \mathcal{O}(\xi n(T_{NN} + T_{RQ}(n, \Delta^d) + \Delta^d + \log n + T_{DEL})) \end{aligned} \tag{8.1}$$

□

8.2.4. Making Things concrete

Elimination Sequences in 2-d

Let us consider a 2-dimensional scenario first. Here we are considerung disks growing in a 2-dimensional plane. This scenario is very closely related to our labeling described in Chapter 5. Given the algorithm from above, the only thing to do is to specify how to answer the spatial queries.

Chan in 2010 [Cha10] gave a theoretical good upper bound on the complexity of the required operations:

Theorem 4 (Chan [Cha10])**.** *For n points in $\mathbb{R}^2$ one can construct a data structure in expected $\mathcal{O}(n \log^2 n)$ time which supports deletions in expected amortized $\mathcal{O}(\log^6 n)$ time, nearest neighbor queries in $\mathcal{O}(\log^2 n)$, and range reporting queries in time $\mathcal{O}(\log^2 n + k \log n)$, where $k = |S|$ is the size of the output of the range reporting query.*

Using this, we get a theoretical bound on the expected amortized running time of our algorithm. In this scenario $\xi = \Delta^2 + 2^5 \in \mathcal{O}(\Delta^2)$.

$$\mathcal{O}(\Delta^2 n(\log^2 n + \Delta^2 \log n + \Delta^2 + \log^6))$$

In practice a Delaunay triangulation performs quite well in providing the required operations. The triangulation can be computed in expected $\mathcal{O}(n \log n)$ [Kre+00]. NearestNeighbor and Range queries can be answered by simply checking the adjacent nodes in the triangulation. See Section 3.3.2 for more details. Delete is possible by removing the point and restoring the delaunay property. Only the very close environment of the point needs to be altered. In the worst case there could be nodes with very high degree making the operations expensive. In practice the maximum node degrees are very low and so the operations can be performed very efficiently [Kre+00].

Elimination Sequences in higher dimensions

Unfortunately, in higher dimensions there is no data structure which supports efficient nearest neighbor and range queries while being dynamic, i.e. allowing to remove points from the data structure efficiently. An approach to bypass this is to use the approximate counterparts (see Section 3.2.3 for a detailed explanation):

Definition 8 (ApproxNearestNeighbor(q))**.** *For a point set $P \subset \mathbb{R}^d$ and a query point q, an ε-approximate nearest neighbor of q is a point $p \in P \setminus \{q\}$ with $|q, p| \leq (1 + \varepsilon) \cdot |q, p'|$, for all $p' \in P \setminus \{q\}$.*

A preliminary version of Section Elimination Sequences in 2-d was published in [FKS16].

A preliminary version of Section Elimination Sequences in higher dimensions was published in [Bah+17].

Definition 9 (ApproxRange(q, r))**.** *For a point set $P \subset \mathbb{R}^d$, a query point q, and a distance r, an ε-approximate range reporting query returns the set S with $S \supseteq \{p \in P \setminus \{q\} : |q, p| \leq r\}$ and $S \subseteq \{p \in P \setminus \{q\} : |q, p| \leq (1 + \varepsilon)r\}$.*

For arbitrary (fixed) dimension, Mount et al. in [MP10] presented a so-called *quadtreap* which allows to answer such queries efficiently. The data structure guarantees the following:

Theorem 5 ([MP10])**.** *Given a set of n points in $\mathbb{R}^d$, a quadtreap storing these points has space $\mathcal{O}(n)$. The tree structure is randomized and, with high probability, it has height $\mathcal{O}(\log n)$. Letting h denote the height of the tree:*

1. *It supports point insertion in time $\mathcal{O}(h)$.*
2. *It supports point deletion in worst-case time $\mathcal{O}(h^2)$ and in expected-case time $\mathcal{O}(h)$ (averaged over all points in the tree).*
3. *Approximate range-reporting queries can be answered in time $\mathcal{O}(h + \left(\frac{1}{\varepsilon}\right)^{d-1} + k)$ where k is the output size.*
4. *Approximate nearest-neighbor queries can be answered in time $\mathcal{O}(h + \left(\frac{1}{\varepsilon}\right)^{d-1})$.*

For our purpose, we do not require very small values for ε in the data structure of [MP10], $\varepsilon = 1$ suffices for our purposes.

To adapt Algorithm 1 to the new approximate spatial queries, we have to change the PredictCollision function, see the modified pseudocode in Algorithm 3.

We first try to find the true nearest neighbor of the current point. If this point is further away than twice the current disk size, we insert an UpdateEvent- hence we need to find the correct one only if it is closer than $2rad(p)t$. Since we only have the approximate range query with $\varepsilon = 1$, we need to query the approximate $4 \cdot rad(p) \cdot t$-range to be sure to have the correct nearest neighbor within the result set. We get the exact $2 \cdot rad(p) \cdot t$-range by removing all those points of larger distance from the approximate result set. If the range was not empty, we determine the point which is closest to p. This is the true nearest neighbor and we proceed just like before.

If this range is empty we search for the approximate nearest neighbor. The collision time with the true nearest neighbor nn is earliest at $|p, nn|/(2rad(p))$ (assuming p is the larger of the collision partners). The found approximate nearest neighbor might be of twice the true nearest neighbor distance, i.e. $|p, ann| \leq 2|p, n|$. Hence we never overestimate the next collision time if we create an UpdateEvent at time $\frac{|p,ann|}{4 \cdot rad(p)}$.

In both cases, the loop invariant as described in Theorem 2 still holds, proving correctness of the modified algorithm.

Algorithm 3: PredictCollisionANN

Data: $\mathsf{P} = \{p_1, \ldots, p_n\}$,
$rad : \mathsf{P} \to \mathbb{R}^+$,
$prio : \mathsf{P} \to \mathbb{N}$ with $prio(p) \neq prio(q)$ if $p \neq q$
Input : Point p, Time t
Output: CollisionEvent (p, q, t) or UpdateEvent (p, p, t)

$anh \leftarrow$ `P.AproxRange (`p, $4 \cdot rad(p) \cdot t$`) ;` `// approx neighborhood`
$nh \leftarrow \{q \in anh : |p, q| \leq 4 \cdot rad(p) \cdot t\}$ **if** $nh \neq \emptyset$ **then**
  $nn \leftarrow \underset{p \in nh}{\operatorname{argmin}}(|p, q|)$; `// neighbor with minimum distance`
  $t' \leftarrow \frac{|p,nn|}{2 \cdot rad(p)}$ **if** $t < t'$ **then**
    return (p, p, t') ; `// UpdateEvent`
  else
    $q \leftarrow \underset{p \in nh}{\operatorname{argmin}}(t_{coll}(p, q))$; `// neighbor with minimum` t `_coll`
    return $(p, q, t_{coll}(p, q))$; `// CollisionEvent`
else
  $ann \leftarrow$ `P.AproxNearestNeighbor (`p`);`
  $t' \leftarrow \frac{|p,ann|}{4 \cdot rad(p)}$ **return** (p, p, t') ; `// UpdateEvent`

To bound the runtime, we make the following observations. The number of calls to *PredictCollision* triggered by a CollisionEvent is still in $O(n)$, with the same argumentation as above. It remains to bound the number of UpdateEvents.

Consider some point p. We show that for each UpdateEvent created for p with the original algorithm, we create at most a constant number of UpdateEvents in the variant using approximate proximity queries. Consider an UpdateEvent popped from Q which leads to a new UpdateEvent. If t was determined by the distance to an approximate nearest neighbor, the correct nearest neighbor had to be within $4 \cdot rad(p) \cdot t$ from p at the moment of creation. Assuming the nearest neighbor did not change, we now find the true nearest neighbor of p in the range query. So we have the same information about p as the original algorithm, and hence will insert the same subsequent events. In case the range is empty, we know that the former nearest neighbor of p has been eliminated meanwhile. But in this case, also the original algorithm would have created a new UpdateEvent for p. Hence, the modified algorithm triggers at most twice the number of UpdateEvents for p as the original algorithm did. Accordingly, the number of UpdateEvents is still in $\mathcal{O}(n)$.

So in fixed dimension d, the number of calls to *PredictCollision* is in $\mathcal{O}(n)$ just like for the original algorithm. The cost of *PredictCollision* is $O(T_{ANN}(n) + T_{ARQ}(n) + \Delta^d)$ where $T_{ANN}(n)$ and $T_{ARQ}(n)$ denote the times for the approximate equivalents of nearest neighbor and range reporting queries. Another $\mathcal{O}(\log n)$ is required for inserting the events into the queue.

With high probability the data structure of Mount et al. ([MP10]) has height $h = \mathcal{O}(\log n)$. Hence, we can handle DELETE in expected $\mathcal{O}(\log n)$, APROXNEARESTNEIGHBOR in $\mathcal{O}(\log n)$ and APROXRANGE in $\mathcal{O}(\log n + k)$ where k is the size of the result set

Let us now plug together the algorithm and the running times of the approximate proximity data structure:

Theorem 6. *In fixed dimension d, we can compute the elimination sequence with high probability in expected time $\mathcal{O}(\Delta^d n(\log n + \Delta^d))$.*

8.2.5. Experimental Results

The previous sections showed how elimination sequences can be computed in two as well as in higher dimensions.

In a practical implementation instead of using planar data (e.g. via Mercator Projection), we used the points located on a virtual "globe". Disks are then aligned and growing along the surface of this "globe". Intuitively, we would use a 3 dimensional problem with spheres centered at our points and compute the corresponding 3d elimination sequence. Unfortunately this is not correct as the spheres will not touch at the surface of the globe but below. Figure 8.6 shows a cut through the globe with two spheres touching.

Instead, the correct solution is to use a $2d$ scenario and replace the euclidean distance function by the great-circle distance. The later can be computed using the Haversine Formula [Wik19c].

For implementing the proximity queries in this -scenario, we used a Delaunay Triangulation of the unit sphere surface $\mathbb{S}^2$. This seems not too hard since its structure resembles exactly the (3d-)convex hull of the set of points. As shown in Section 8.2.4, the required queries can be answered by simple graph traversal in the Delaunay graph. While we were not aware of a ready-to-use spherical Delaunay triangulation package, we could adapt the CGAL 2d-Delaunay triangulation data structure ([Pro15]).

We interpret the 2-dimensional coordinates of the planar Delaunay triangulation as spherical coordinates. All required predicates of the Delaunay triangulation are adapted, to match the spherical scenario. One main obstacles of this approach is the localization in the triangulation. We had to synthesize a suitable initial

A preliminary version of Section 8.2.5 was published in [Bah+17].

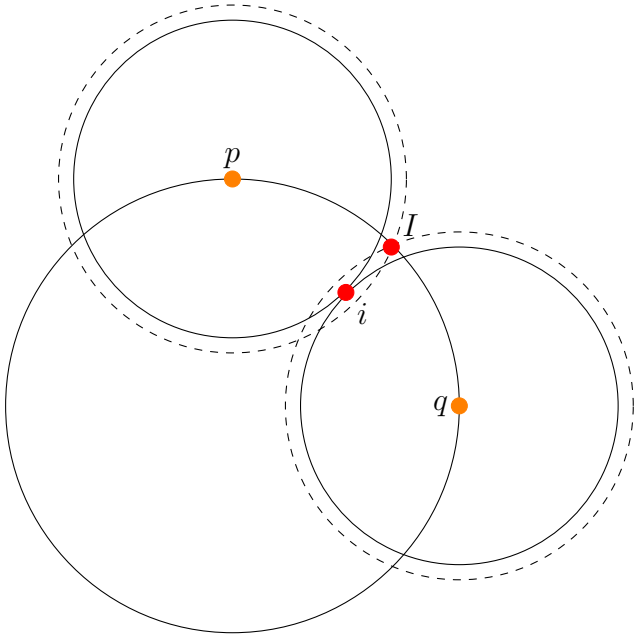

Figure 8.6.: Cut through a sphere with two points p, q. The intersection i of the spheres induced by p and q (solid line) is below the sphere surface. The intersection along the surface I is at a larger radius for p and q.

triangulation: one very small triangle around the north pole and a node at the south pole connected to all three nodes of the first triangle. The main predicate for maintaining the Delaunay triangulation is the following: for a given query point s and a triple of points p, q, r check if s is within the circle defined by p, q, r or not. In the given scenario this predicate can be implemented as a $3d$ orientation tests. The three points are defining a plane in $3d$. Now s is closer to the sphere center if and only if s is not contained in the circle defined by the points p, q, r. See 8.7 for an illustration.

The given implementation of the *in_circle*-test depends strongly on the fact that the mapping from the $2d$-spherical coordinates to the $3d$ coordinates is correct. If the point in $3d$ is not exactly located on the unit sphere, the predicate delivers false results. This may lead to an incorrect Delaunay triangulation. Daniel Bahrdt and Martin Seybold in [Bah+17] presented how to solve this issue by finding rational points exactly on the sphere.

We implemented this Delaunay-based spatial datastructure using the CGAL library [Pro15]. Technically, the adaptation is performed by creating a triangulation class and a CGAL traits class which is derived from a CGAL kernel. The triangulation class inserts the initial triangulation, while the traits class provides the necessary predicates for the in-circle test, the orientation test, and distance

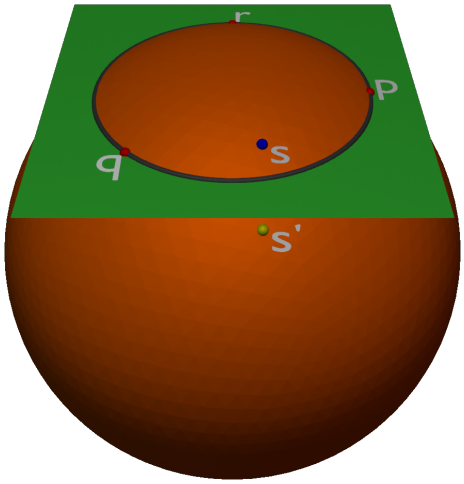

Figure 8.7.: The *in_circle*-test can be done via $3d$ orientation test relative to the pqr-plane. Point s, which is within the circle, is on the opposite site than the sphere center. The point s', which is outside of the circle is at the same side than the sphere center.

comparison which have to be performed with the setting on $\mathbb{S}^2$ in mind. Running times were measured on a single core of an AMD Ryzen 7 2700 and 32GB of RAM. The output of our algorithm is the elimination sequence in which the items disappear over time together with the collision partner responsible for their elimination. We implemented an OpenGL Viewer to visualize the result. It allows for smooth zooming and rotation of the globe, as well as continuous variation of time. For two screenshots of this visualization see 8.8 and 8.9.

Synthetic Benchmarks

size	time (s) storage	time (s) events	space (MB)
10^3	0.035	0.029	7.116
10^4	0.344	0.199	17.924
10^5	3.415	2.267	119.176
10^6	34.382	27.991	1,037.704
10^7	347.292	334.951	11,412.196

Table 8.1.: Running times and peak space (RAM) consumption for synthetic data.

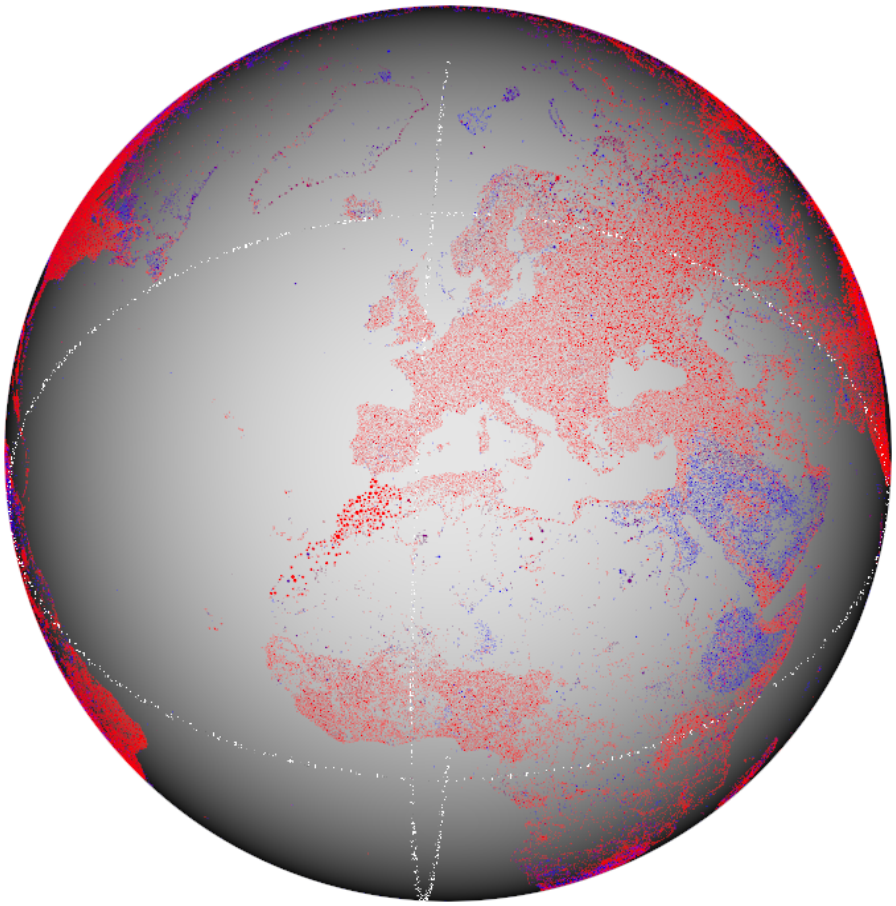

Figure 8.8.: Visualization of the globe at a specific t. The dots are corresponding to the growing disks on the surface each representing a displayed label.

First, we generated the amount of random points on the sphere. To have them evenly distributed of the sphere surface, we used the tool contained in the Rational Points on Sphere library [BS17]. We assigned a random radius and a unique priority to each point. In table 8.1, we report running times for the initial Delaunay Triangulation construction and the actual algorithm. We also report the peak memory consumption. The running time of both, the Delaunay triangulation construction as well as the event processing is nearly linear. With more main memory (e.g. $256GB$ of RAM), we could easily compute an elimination order for half a billion points in the order of a few hours.

Real World Benchmarks

For real-world instances, we use data from the OpenStreetMap project [Con17c]. We extracted all nodes with a *name*-tag. Additionally, we extracted nodes representing ATMs, cafés, restaurants and many more. These were represented by an icon if they did not have a specified name. We prioritize settlements over other POIs. Within the settlements, cities are prioritized to towns etc. In the POI group, we defined an ordering on the POI types. Settlements where further subclassified using population sizes (if available) to further refine the priorities. To get the required total order, we sorted elements in the same group according to their osm id. For each item, the radius of the corresponding sphere was computed as $r = \frac{|label|}{2}$. Here $|label|$ is the size of the label in a fixed font with a font size depending on the importance level of the POI.

The figures 8.8 and 8.9 both are based on the planet data set.

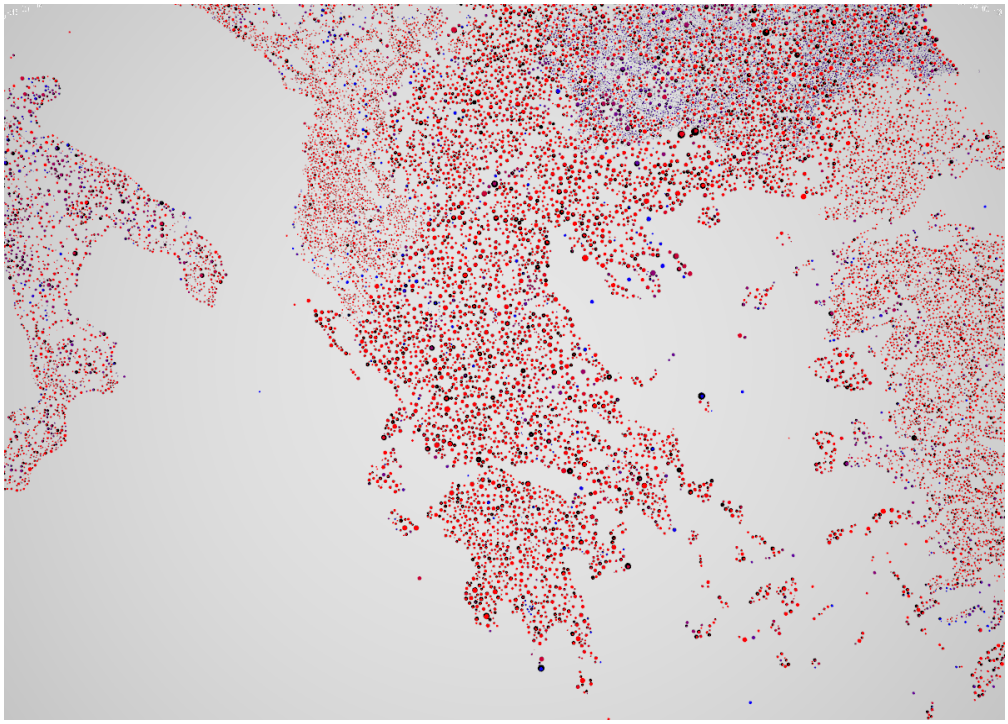

Figure 8.9.: Visualization of the growing disks of Greece. The dots are corresponding to the growing disks on the surface each representing a displayed label.

Having derived location, radius and priority for all the POIs, we ran our algorithm to determine an elimination order. In Table 8.2, we have performed the same measurements we did for the synthetic data. In about 15:24 minutes, we can

process the majority of the items of the Planet data set using our algorithm.

data set	size	time (mm:ss) storage	time (mm:ss) events	space (MB)
Stuttgart	82k	0:03	0:02	105
Ba-Wü.	248k	0:08	0:07	269
Germany	1.4m	0:47	0:42	1,584
Europe	7.7m	4:12	4:27	7,780
Planet	13.3m	7:41	7:44	13,992

Table 8.2.: Running times of the storage initialization and the event handling. Data has been extracted from the OpenStreetMap project. The data set sizes are given in thousands (k) and millions (m).

8.3. Elimination Sequences with Unrestricted Priorities

Lets go back to the original scenario, where the priority function does not have any restrictions. In Section 8.3.1, we introduce a Mixed-Integer linear program (MLP) to solve this scenario. The reader can find a very short introduction to linear programs in Section 3.3.3. The so computed solution is optimal but requires a large computation power and RAM. Solving an instance of $1,000$ points requires about 200 gigabytes of RAM and several days to compute.

However, we might use our introduced algorithm to compute an approximated solution. In Section 8.3.2, we describe several heuristics allowing to break ties - i.e. define strategies to resolve collisions of points of equal priority. Thus computing an approximation of the optimal result with a much lower computation power and time consumption. In Section 8.3.3, we compare the quality of the approximated result with the optimal result obtained by the MLP.

8.3.1. A Mixed-Integer Linear Program Formulation

We introduce some **LP Variables** as follows:

$\forall p \in P : et_p \in \mathbb{R}$ indicates the elimination time of point p

For each tuple of points $(p, q) \in P \times P, p \neq q$, we introduce a binary variable:

$$res_{pq} = \begin{cases} 1 & \text{if } p \text{ resolves the conflict. This implies: } et_p \leq t_{coll}(p, q) \\ 0 & otherwise \end{cases}$$

In the following let $p \neq q \in P$ be an arbitrary tuple of distinct points with $t_{coll}(p, q) \geq popup(p)$ and $t_{coll}(p, q) \geq popup(q)$. The following set of constraints need to be fulfilled.

1. Exactly one of the two needs to resolve the conflict:

$$res_{pq} + res_{qp} = 1$$

2. If p resolves the conflict, its elimination time is less (or equal) than the collision between the two points:

$$et_p \leq t_{coll}(p, q) + res_{qp} \cdot t_{max}$$

   and vice versa.

3. If p resolves a conflict with q and $prio(p) \leq prio(q)$, q must not have resolved a conflict earlier, i.e. must still be alive at that point in time.

4. If $prio(p) < prio(q)$, p must resolve the conflict with q. If there is an r with $t_{coll}(r, q) \leq t_{coll}(p, q)$ and $popup(r), popup(q) \leq t_{coll}(p, r)$ and q resolves this conflict with r, q can resolve the conflict with p:

$$res_{pq} + \sum_{r \in C} res_{qr} \geq 1$$

   with $C = \{r \in P | r \neq p \wedge r \neq q \wedge t_{coll}(r, q) \leq t_{coll}(p, q)\}$:

5. And obviously, for each point $p \in P$ the elimination time must be less or equal to its maximum elimination time:

$$et_p \leq et_{MAX}(p)$$

Our **Objective** is to maximize the sum of all elimination times:

$$\sum_{p \in P} et_p$$

The number of variables in the MLP is:

$$\begin{aligned} \#Vars_{BIN} &= \binom{|P|}{2} * 2 \\ \#Vars_{REAL} &= |P| \\ \#Vars &= \#Vars_{BIN} + \#Vars_{REAL} && \in \mathcal{O}(|P|^2) \end{aligned}$$

where the first summand ($Vars_{BIN}$) is the number of binary decision variables and the second one ($Vars_{REAL}$) is the number of et_p variables.

The number of constraints is:

$$\#Constr \leq \binom{|P|}{2} + 2 * \binom{|P|}{2} + 2 * \frac{|P|^2}{2} + |P| \qquad \in \mathcal{O}(|P|^2)$$

Here the first summand is the number of pairwise constraints in 1). The second is the number of ordered tuples used in 2). The third summand is an upper bound to the sum of the number of points with a larger priority for each point. This upper bounds the number of constraints described in 3) and 4). The worst case is that the priority function is chosen like in the simple scenario, i.e. it is a total order. Then the number of constraints is: $\sum_{i=0}^{|P|-1} i = \frac{|P|(|P|-1)}{2}$. The last summand counts the constraints ensuring that the elimination time of each point is less or equals to the specific et_{MAX} value.

8.3.2. Heuristics for Solving the Unrestricted Model

We can solve the unrestricted model using the algorithm presented in Sections 8.2 and 8.2.4. The only difference in the unrestricted model is that we might get to the point where the circles of two points of the same priority are colliding. In this case we are free to choose which to eliminate. This decision affects the upcoming eliminations, hence the value of the solution. If aiming for an optimal solution, we would need to evaluate each of the two possible decisions. Which might require to compute the full sequence for both cases - hence increasing the computational complexity.

A possible solution is to use heuristics for the decision. This would probably not give an optimal result but hopefully a good approximation. In the following we will introduce three heuristics. Their only goal is to decide: Given two points of the same priority, which of the two is eliminated in a collision.

Pseudo-Random (PSR) In this heuristic, we just randomly pick one of the two points. We call this pseudo-random, because in the experiments, we always choose the one of smaller osm id to be eliminated. Using these ids as random source allow us to make the random decisions reproducible.

Radius (RAD) The radius heuristic chooses the point with larger radius for elimination. The remaining point is of lower radius and is growing slower therefore. The underlying idea is that by minimizing the radii of surviving points, leads to good results in our objective.

Next Elimination (NXEx) The idea of the next elimination heuristic is to find the nearest point of larger priority for each of the two points. Since this point will probably eliminate the considered point, we decide based on their collision time. We eliminate the point with a smaller of the two collision

times. Depending on how far we look we may or may not find such a point. In the work at hand we used three parametrizations: NXE2, NXE4 and NXE8 are inspecting a range of 2×, 4× and 8× the size of the current disk radius.

For the introduced heuristics, there might be ties occuring, e.g. if the two radii are equal. In these cases, we fall back to our Pseudo-Random heuristic to break the ties according to the osm id.

8.3.3. Evaluation

For the evaluation, we used three data sets and compute their optimal solutions with our LP formulation. The *Bremen* data set contained the names of 142 residential subareas of the city of Bremen. The labels were classified into 6 priority levels. The *BW_1k* data set contained the names of the $1,000$ most important residential areas in Baden-Württemberg. The *S_1k* data set contained the names of the $1,000$ most important residential areas in the government district of Stuttgart. The data sets were retrieved from the OpenStreetMap data set. The level sizes of the priority levels can be found in table 8.3. The benchmarking of the LPs was done on a server with two Intel(R) Xeon(R) CPU E5-2650v4 with 24 cores in total and 768 GB of RAM. While the LP construction was done single threaded, the LP solution was computed using Gurobi with 24 threads. In table 8.3, you see time and space consumption for solving the LP. You can easily see the enormous time and space consumption even for the small data set of $1,000$ elements. Especially for the Stuttgart data set, the required 4 days, just for approximating the optimum solution to about 25% are noticeable.

data set	#items	Prio level sizes	Constr (mm:ss)	Solving (mm:ss)	RAM RAM	Quality Quality
Bremen	142	$(1, 1, 33, 5, 90, 12)$	$0:13$	$0:02$	561MB	100%
BW_1k	$1,000$	$(1, 8, 61, 268, 662)$	$78:15$	$39:54$	185GB	100%
S_1k	$1,000$	$(1, 1, 25, 84, 889)$	$81:47$	$> 4days$	250GB	25%

Table 8.3.: Three benchmark data sets with the number of items and items in each priority level (left is most important). Time consumption for construction (single threaded) and solving the LP (24 threads) and overall required RAM. The Quality column indicates how close the result is to the optimum (100% is exact).

In table 8.4, you see the optimal results of the *Bremen* data set in column OPT. The next columns show the values of the heuristics, we defined in 8.3.2, ordered by their solution value. The solution values are given in total and relative to the

optimum in percent. In the last two rows the results of the inverted heuristics are given. What means that we always eliminate the one point which should not be eliminated according to the corresponding heuristic.

	OPT	NXE4	NXE8	RAD	NXE2	PSR
Bremen	$410k$	$395k$	$395k$	$383k$	$370k$	$363k$
	100%	96.3%	96.3%	93.4%	90.2%	88.5%
inverted		$367k$	$367k$	$344k$	$351k$	$363k$
heuristic		89.5%	89.5%	83.9%	85.6%	88.5%

Table 8.4.: Values of the optimal solution of the Bremen data set and the heuristics as defined in Section 8.3.2. The heuristics are sorted in an descending order from left to right. The second row (inv heur) contains the solution if always the inverse decision of the heuristic is taken.

Table 8.5 shows the same results, but for the *BW_1k* data set.

	OPT	RAD	NXE4	NXE8	NXE2	PSR
BW (part)	$6,586k$	$6,203k$	$6,147k$	$6,140k$	$5,934k$	$5,750k$
	100%	94.2%	93.3%	93.2%	90.1%	87.3%
inverted		$5,290k$	$5,442k$	$5,401k$	$5,605k$	$5,681k$
heuristic		80.3%	82.6%	82.0%	85.1%	86.3%

Table 8.5.: Values of the optimal solution of the BW_1k data set and the heuristics as defined in Section 8.3.2. The heuristics are sorted in an descending order from left to right. The second row (inv heur) contains the solution if always the inverse decision of the heuristic is taken.

The first thing you see is the huge difference in the time and space consumption of the optimal LP solutions compared to our algorithm. While the LP of a thousand points requires nearly 2 hours and 200GBs of RAM, we can compute elimination sequences of 13 millions of points in around 15 minutes on a standard desktop computer with our algorithm. Using the simplest of the heuristics, the RAD heuristic gives us a good approximation ratio of only 5% below the optimum.

		Stuttgart	Germany	Planet
	# items	81994	1413594	13302969
	max($\|lvl\|$)	22112	391250	1331742
PSR	value (%)	82007k(97%)	1602200k(96%)	35764987k(91%)
	time (mm:ss)	0 : 05	1 : 36	15 : 54
RAD	value (%)	84310k(100%)	1669029k(100%)	39281151k(100%)
	time (mm:ss)	0 : 05	1 : 38	15 : 56
NXE2	value (%)	81622k(97%)	1599468k(96%)	35780431k(91%)
	time (mm:ss)	0 : 05	1 : 39	16 : 09
NXE4	value (%)	81594k(97%)	1607921k(96%)	36359776k(93%)
	time (mm:ss)	0 : 05	1 : 40	16 : 37
NXE8	value (%)	83118k(99%)	1623164k(97%)	37200808k(95%)
	time (mm:ss)	0 : 06	1 : 44	18 : 00

Table 8.6.: Comparison of runtime and qualities of the different heuristics for three data sets. The total number of items and the size of the largest priority class are given. The value percentages are compared to the best performing heuristic.

Part IV.

Epilogue

Conclusion

Dynamic maps, allowing to pan, rotate and zoom the map view continuously, are widespread. In contrast to maps with discrete zoomlevels or even paper maps with a fix map scale, these maps have special requirements to the map labeling. Map labeling is subdivided into three domains: The labeling of area-, line- and point-like features. In the work at hand, we have considered two of these three domains.

Area labels are supposed to be fit into the area and represent the overall area shape. Therefore, they are allowed to be bend along a circular arc. In our area labeling scheme, we approximate the area label by a box of the same length-to-height aspect ratio. We search for an optimal placement of the box within the area, hereby allowing the box to be bend along a circular arc. We introduced an algorithm to find such a placement efficiently. The algorithm approximates the area skeleton. For each edge of the skeleton, the smallest distance to the area boundary is determined. Based on this information we search for promising paths in the skeleton and approximate these paths with a circular arc. A set of candidate arcs is evaluated, i.e. an optimal placement of the box along that arc is determined. The optimum of these placements is returned as the area label position.

The efficient computation of the label position, allows for computing area labels in near-real time. Hence making it possible to label areas, not fully contained in a map view, dynamically. Thus, the viewer can always be presented with a label - even if only a very small part of an area is reaching into the visible area. Considering map zoom, an optimal label box can be used to determine the map scale at which the area should be refined or replaced by a containing entity. Refining could be done by labeling subareas, f.e. the city districts in case of the area being a city. On the contrary, the area label of a city could be replaced by the label of the surrounding administrative area.

Point labels are horizontally aligned and are visually ”touching” the corresponding point in order to allow a clear attribution. Since the labels are kept at constant sizes, the number of labeled entities need to decrease when zooming out of the map. When rotating the map, the label set must not change in order to present a constant view to the user. We therefore propose to reserve a so called label disk. A disk which fully contains the label in any possible rotation. By choosing this

set of disks to be non-overlapping, a constant labeling on rotation can be guaranteed. The disk size is variable and inversely correlated with the map scale. So the label disks are growing - relative to the map - when zooming out. Accordingly the label sizes grow. Whenever during a continuous zooming out operation, two disks touch, the one corresponding to a less important point needs to be removed. The corresponding label is removed to keep the labeling free of label-intersections. Considering the process, starting at a very large map scale. Initially, all the label disks are non overlap ing since their radius tends to 0. Decreasing the map scale leads to the disks growing and eliminating each other. They so define a *elimination sequence* of the points and their corresponding elimination-map-scale.

We are considering the generalized, d-dimensional elimination sequence problem separately. For this problem we show $NP-completeness$ and provide an simplified scenario where the sequence is unique. An algorithm is presented which compute the elimination sequence efficiently. In the general d-dimensional case with high probability in expected $\mathcal{O}(\Delta^d n(\log n + \Delta^2))$. In the 2-dimensional case in expected amortized $\mathcal{O}(\Delta^2)n(\log^6 n + \log^2 n + \Delta^2 \log n + \Delta^2)$. Where Δ is the ratio between the maximum and minimum radius in the instance i.e. $\Delta = \frac{r_{max}}{r_{min}}$. These theoretical bounds are depending on sophisticated data structures for answering spatial queries. In practice the algorithm was implemented using a Delaunay-based data structure. With this implementation we could compute elimination sequences of 13 million points in about 15 minutes.

While these results are optimal for the simplified scenario, we used a linear program to compute optimal solutions to the original problem. We could compute an optimum for one instance of 142 points and another for an instance of 1.000 points. Approximations of solutions to the original problem could be computed with the presented algorithm using heuristics to break ties. For the instances, where optimum solutions could be computed, the best heuristic got an approximation rate of about 96%. Using these heuristics, we could compute approximate solutions even for data sets of millions of items.

Further Research

The intensive research over many years has yielded many results as shown in the work at hand. Nevertheless, it also revealed many issues which could not be completed to satisfaction. For some of the issues described in the following there are drafts for solutions. Others require to put large effort into implementation or performing user studies.

The later is the case when it comes to evaluating the research results. Currently, the approach presented in the work at hand is based on one main assumption: In interactive maps, a decreased level of detail of a map view can be compensated by

the available map zooming operation. Based on this assumption, we conclude that reducing the level of detail in favour of consistency during zoom and rotation is acceptable. At least in scenarios like navigation this consistency is supposed to be preferred over a more dense labeling. Now that the alternative approach described in the work at hand is available, these assumptions should be evaluated in user studies. In order to do so, a lot of effort has to be put into a visual implementation. Until now there have been two student projects working on an adaptation of the well known OpenLayers web-framework. Unfortunately the results do have performance issues since a lot of unnecessary client-server communication is induced.

A very promissing issue is the combination of area and point labels. The presented area labeling approach allows for determining a map scale at which an area label can no longer be presented properly. During continuous map zooming this area label might seamlessly be replaced by a point label. But this requires to intertwine area and point label data sets - which are currently separated. Nevertheless with the area labeling approach and the concept of popup times for point labels, the required concepts are available.

The experimental results of the area label computation revealed a good performance of the labeling scheme - even without simplified polygons. By simplifying the polygon, e.g. with the Douglas-Peucker algorithm [DP73], the performance could be further improved by decreasing the input size. It should be possible to decrease the input polygons by a factor of 10 without any loss of quality. However, one has to outbalance the subsampling and the polygon segment length. Longer polygon segments increase the inaccuracy of the wedge-based placement search procedure. For the label quality a meaningful choice of candidate paths is crucial. Our proposed scheme for finding a candidate set performs much better than a naive selection of the 50 longest paths. Still we think that this selection can be further improved. A thesis focusing this topic is currently work in process. Despite the candidate path selection, the proposed method for computing the optimal box placement along the baseline is a major result of the work at hand. It might be further improved by taking into account the perceived coverage presented by Barrault. Especially when the label box is restricted by the boundary polygon in his height, stretching or shifting the label along the baseline might further increase its perceived coverage. This could lead to a better overall labeling quality.

The presented point labeling scheme is the most elaborated concept and hence the main result of the work at hand. As stated above, it remains to evaluate the concept in user studies. The main weakness of the approach is the reduced level of detail in a map labeled with presented approach. But the presented extensions like popup times and multilayer labelings provide solutions to this issue. The concept of elimination sequences inherently provide a property which allows to

reduce the amount of requests for labelings during zoom. A labeling for a specific map scale is always a subset of the labeling for a larger map scale. Which means that a zooming out of a map view can be implemented without requesting a new labeling. The only thing required is to filter the available labels with the new map scale. Panning the map view also does not affect the current labeling - except for labels sliding in and out of the map view. The former can be implemented by just querying the newly visible area, the later by just dropping invisible labels. These properties can be used to implement efficient caching to minimize the requests and required communication in an client-server based visualization.

The work at hand omits the labeling of line features. Including these into the labeling scheme is a natural next step. Several links come into mind. First area features may turn into line features when zooming out - think of rivers. Accordingly the area labels should turn into line labels on smaller map scales. Further zooming out might reduce the line feature to a single point and a corresponding point label. Elimination times should be a good way to model this process.

Part V.

Appendix

Code Projects

During the work on the research described above, various software projects were developed. The projects range from creating labeling data sets from the OpenStreetMap data over cleaning up the data sets to actually (pre-) computing the labeling. We designed the project for OpenStreetMap

The projects are all OpenSource and publicly available at the VGIScience code repository here https://gitlab.vgiscience.de/map_labeling. An overview of these projects is given in the following sections.

Section Area Labeling contains projects related to the labeling of areas. The following Section Point Labeling covers the projects dealing with the labeling of points.

Area Labeling

Data Generation

In the context of map labeling, a very intuitive input for map labeling is the administrative areas. OpenStreetMap provides this information in their data set. The areas are classified into several levels, where the national border is the highest. In Germany the next level is the federal states (i.e. Bundesländer) followed by the state-district (i.e. Regierungsbezirke) and county level (i.e. Landkreise). Five additional levels define lower ranked urban, suburban and rural areas (for more details see [Wik19b; Wik19a]).

The **osm_area_extractor_rs** is a project writen in Rust to extract administrative areas from a given OpenStreetMap pbf file. By providing a minimum level, the result can be restricted to administrative levels of higher levels only. The project is published here: https://gitlab.vgiscience.de/map_labeling/area_labeling/osm_area_extractor_rs. A preliminary version, implemented in C++, is hosted here: https://theogit.fmi.uni-stuttgart.de/geodata-visualization-project/area_extractor. This project additionally allows to extract points with a matching administrative level as well as incomplete areas. The later are areas for which information, like boundary coordinates, are not provided by the pbf file.

Data Cleaning

The data contained in the OpenStreetMap data set might be dirty, e.g. by containing areas with self-intersections or holes of an area intersecting the area boundary. To clean-up the data and provide proper input for projects building uppon, the **poylgonize** project. For an input collection of segments and points in $2d$, a representation with polylines and polygons is created. The resulting polygons do not contain self intersections and inclusion-relations are guaranteed to be correct. The project is joint work with Thomas Mendel and is hosted at the VGI-Science code repository: https://gitlab.vgiscience.de/map_labeling/area_labeling/polygonize.

Label Computation

The area label computation, as described in Chapter 7, is implemented in the **area_labeling** project. The project was joint work with Thomas Mendel. The sourcecode is published here: https://gitlab.vgiscience.de/map_labeling/area_labeling/area_labeling. Amongst others, a library is provided to compute the label position for a given polygon. A mockup jupyter-notebook is contained for creating a quick illustration.

Point Labeling

Data Generation

Generating labeling data for point labelings with the labeling scheme presented in Chapter 5 consists of essentially two steps. First, one is after finding the set of points to label. Second, these points one need to be classified to determine their priority, labeling parameters like font size or an icon (and its size) which should be used as label. In the OpenStreetMap data set, nodes are candidates for a point label. Whether a node actually qualifies for being labeled, can be determined by considering their tag set. Depending on these tags, the nodes can be classified and their priority and labeling parameters determined. The **osm_input_rs** project is a Rust implementation of an input generator. The project is hosted at https://gitlab.vgiscience.de/map_labeling/poi_labeling/osm-input-rs. A preliminary version, implemented in C++, is available at https://theogit.fmi.uni-stuttgart.de/maplabeling/osm_input. Both are using an json configuration file to define the available poi classes, priorities and presentation parameters.

Label Computation

Precomputing a label elimination sequence is implemented in the **growing_balls** project. The project is published at the VGIScience repository: https://gitlab.vgiscience.de/map_labeling/poi_labeling/growing_balls. A meta project, the **label_pipeline**, combines osm_input and elimination sequence computation. It allows to compute an elimination sequence directly from a pbf file and uses the above described projects as submodules. The label_pipeline is hosted at the VGIScience repository here: https://gitlab.vgiscience.de/map_labeling/poi_labeling/label_pipeline.

Label Querying

As described in Chapter 5, a concrete labeling for a given map scale can be obtained by filtering the precomputed elimination sequence. In a realistic scenario a second filter needs to be applied: to spatially restrict the result set to the current map view. The **PS2dT** project, available at https://gitlab.vgiscience.de/map_labeling/poi_labeling/ps2dt, implements a 2-dimensional priority search tree in Rust. This essentially is a priority search tree combined with a 2-dimensional kd-tree. The data structure allows to efficiently query a consistent labeling for a given bounding box and map scale.

Zusammenfassung

Navigationssysteme, beziehungsweise digitale Kartenapps, haben sich zu alltäglichen Begleitern entwickelt. Waren digitale Karten vor einem Jahrzehnt ein wertvolles Gut - Updates für Navigationssysteme wurden für viel Geld verkauft - sind sie heute durch die Verbreitung von Google Maps und Co. eine Selbstverständlichkeit. Neben der Verfügbarkeit und der Möglichkeit einfacher Updates, ermöglichen digitale Karten eine einfache, intuitive Interaktion. Das Zoomen, Verschieben und Drehen des Kartenausschnitts zum Erkunden von Details und der Ausrichtung in Blickrichtung des Nutzers, sind wohlbekannte Funktionen. Letzteres ist vor allem bei Navigationssystemen zentral. Dort wird die Kartenausrichtung generell in Fortbewegungsrichtung gewählt. Insbesondere für die Darstellung von Kartenbeschriftung, stellen diese Möglichkeiten eine Herausforderung dar.

In der vorliegende Arbeit wird ein Konzept zur Gestaltung von Beschriftungen in interaktiven Karten, die stufenloses Zoomen, Verschieben und Rotieren des Kartenausschnitts ermöglichen, vorgestellt. Dabei wird insbesondere die Beschriftung von sogenannten "Points of Interest" - also zu beschriftenden Punkten - thematisiert. Derartige Beschriftungen sind gewöhnlich horizontal ausgerichtet und überlappen sich somit tendenziell bei der Kartenrotation. Vor allem beim Zoomen der Karte sind verschiedene Konsistenzkriterien zu beachten - so soll beispielsweise eine Beschriftung während eines kontinuierlichen Zoomens nicht mehrfach hinzugefügt und wieder entfernt werden. Außerdem sollen Beschriftungen von wichtigeren Punkten länger sichtbar sein, als solche von weniger wichtigen Punkten.

Bei der Beschriftung von Gebieten gilt, dass sich die Beschriftung im Gebiet befinden und die grobe Form des Gebiets adaptieren soll. Um letzteres zu erreichen, darf die Beschriftung entlang eines Kreisbogens gebogen sein. Rotation ist in diesem Szenario einfach - die Beschriftung muss lediglich in ihrer Ausrightung umgekehrt werden, sodass sie nicht auf dem Kopf steht. Beim Zoomen des Kartenausschnitts sollten Gebiete als Ganzes oder ihre Untergebiete beschriftet werden, abhängig von der Kartenskala der Gebietsbeschriftung.

Für der Beschriftung von Punkten werden sogenannte "Disk-Label" eingeführt. Bei diesen wird für jede Beschriftung ein kreisförmiger Bereich um den zu beschriftenden Punkt reserviert. Die Beschriftung des Punkts bleibt in beliebigen Ausrichtungen der Karte vollständig in diesem Bereich enthalten. Indem als Beschriftung in einer beliebigen Kartenskala eine überschneidungsfreie Untermenge der "Disk-Label" gewählt wird, ist die lesbare Darstellung bei Rotation gewährleistet. Beim Herauszoomen aus dem Kartenausschnitt, werden die Beschriftungen bei konstanter Größe beibehalten. Entsprechend wachsen die zugehörigen Disks relativ zur Karte - entsprechend schrumpfen die Distanzen zwischen den Disks. Bei fortschreitendem Zoomen, kommt es zu Berührungen der Disks. Dabei muss eine der beteiligten Beschriftungen muss entfernt, um bei weiterem Zoomen die überschneidungsfreiheit zu erhalten. Wird der entsprechende Zoomprozess von "unendlich

weit hineingezoomt" zu "unendlich weit herausgezoomt" betrachtet, ergibt sich eine sogenannte Eliminationssequenz der Beschriftungen. Diese kann für einen Datensatz im Voraus berechnet werden und ermöglicht dann eine einfache, effiziente Ableitung einer Kartenbeschriftung für ein gegebenes Darstellungszenario.

Das Problem der Berechnung der Eliminationssequenzen kann verallgemeinert in einem d-dimensionalen Hyperraum betrachtet werden. Entsprechend der Disks in $2D$, wachsen d-dimensionalen Hyperbälle. Eine Eiminationssequenz ist dabei nicht eindeutig - kollidieren Bälle der selben Wichtigkeit, ist die Entscheidung welcher der beiden Bälle eliminiert wird, essentiell für den weiteren Verlauf der Sequenz. Das Ziel ist die Berechnung einer optimalen Sequenz, in der die Summe aller Eliminationszeitpunkte maximiert wird. Es wird gezeit, dass die Berechnung einer solchen optimalen Sequenz $NP - hart$ ist. Für ein vereinfachtes Szenario - in welchem die Sequenz eindeutig ist - wird ein effizientes Berechnungsverfahren vorgestellt und analysiert. Dieses Verfahren hängt stark von einer effizienten Methode zur Berechnung des nächsten Nachbarn sowie der Suche nach Punkten mit einer maximalen Distanz zu einem gegebenen Punkt ab. Je nach Szenario können diese Operationen effizienter oder weniger effizient gelöst werden. Für $2D$ und für höhere Dimmensionen werden zwei verschiedene theoretische Laufzeitschranken bewiesen. Für die praktische Beschriftung von digitalen Karten, werden Punkte auf einem virtuellen Globus betrachtet. Eine Berechnung der Sequenz für 13 Millionen Punkte ist auf einem herkömmlichen Computer in etwa 15 Minuten möglich.

Das ursprüngliche, unvereinfachte Problem kann mittels eines Mixed-Integer Linearen Programms für wenige Punkte exakt gelöst werden. Allerdings nur mit erheblichem Rechen- und Speicheraufwand. Eine gute Approximation der optimalen Sequenz bieten Heuristiken. Mittels dieser kann der vorgestellte Algorithmus für das vereinfachte Szenario Lösungen für das unvereinfachte Problem berechnen. In der Güte liegen die Heuristiken bei $5\% - 10\%$ unter dem Optimum. Bei Berechnungszeiten von $15 - 20$ Minuten auf einem herkömmlichen Computer.

Eine Gebietsbeschriftung liegt vollständig innerhalb des Gebiets und ahmt die Form des Gebiets nach, indem sie entlang eines Kreisbogens plaziert ist. Für eine gute Beschriftung ist eine entsprechende Platzierung gesucht, welche die Größe der Beschriftung maximiert. Abhängig davon kann eine maximale und minimale Kartenskala berechnet werden, bei welcher die Beschriftung lesbar dargestellt werden kann. Entsprechend kann das Gebiet bei größeren Skalen in einer weiteren Verfeinerung dargestellt, oder bei kleineren Skalen durch eine Punkt-Beschriftung oder die Darstellung eines umfassenden Gebiets ersetzt werden.

Für die Berechnung einer solchen guten Beschriftung wird in der vorliegenden Arbeit ein effizienter Algorithmus vorgestellt. Dieser basiert auf einem Algorithmus von 2001, von M. Barrault in [Mat01] publiziert. Durch verschiedene Optimierungen kann diese Berechnung auf Quasi-Echtzeit verbessert werden.

Bibliography

[AFL84] J. Ahn, H. Freeman, and Image Processing Laboratory. "A program for automatic name placement". In: 1984. URL: https://core.ac.uk/display/21214718 (visited on 01/15/2020).

[Aut] DLMF Authors. *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.24 of 2019-09-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds. URL: http://dlmf.nist.gov/.

[Bah+17] D. Bahrdt et al. "Growing Balls in $\mathbb{R}^d$". In: *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. 0 vols. Proceedings. Society for Industrial and Applied Mathematics, Jan. 1, 2017, pp. 247–258. DOI: 10.1137/1.9781611974768.20. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611974768.20 (visited on 01/04/2018).

[Bau19] Lukas Baur. "Points of Interest - A search for adequate map labellings". Bachelorthesis. Stuttgart: University of Stuttgart, May 19, 2019.

[BDY06] K. Been, E. Daiches, and C. Yap. "Dynamic Map Labeling". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pp. 773–780. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.136.

[Bee+10] Ken Been et al. "Optimizing active ranges for consistent dynamic map labeling". In: *Computational Geometry*. Special Issue on 24th Annual Symposium on Computational Geometry (SoCG'08) 43.3 (Apr. 1, 2010), pp. 312–328. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2009.03.006. URL: http://www.sciencedirect.com/science/article/pii/S0925772109000649 (visited on 01/18/2018).

[Blu67] Harry Blum.
"A transformation for extracting new descriptions of shape".
In: *Models for the perception of speech and visual form* (1967), pp. 362–380. URL: https://ci.nii.ac.jp/naid/10009398958/en/.

[Boo19] Boost. *Boost C++ Libraries*. [Online; accessed 20-Dec-2019]. 2019.

[Bra94] J. W. Brandt. "Convergence and Continuity Criteria for Discrete Approximations of the Continuous Planar Skeleton".
In: *CVGIP: Image Understanding* 59.1 (Jan. 1, 1994), pp. 116–124.
ISSN: 1049-9660. DOI: 10.1006/ciun.1994.1007.
URL: http://www.sciencedirect.com/science/article/pii/S1049966084710072 (visited on 04/24/2019).

[BS17] Daniel Bahrdt and Martin P. Seybold. "Rational Points on the Unit Sphere: Approximation Complexity and Practical Constructions".
In: *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '17.
event-place: Kaiserslautern, Germany.
New York, NY, USA: ACM, 2017, pp. 29–36.
ISBN: 978-1-4503-5064-8. DOI: 10.1145/3087604.3087639.
URL: http://doi.acm.org/10.1145/3087604.3087639 (visited on 08/06/2019).

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. "Unit disk graphs".
In: *Discrete Mathematics* 86.1 (Dec. 14, 1990), pp. 165–177.
ISSN: 0012-365X. DOI: 10.1016/0012-365X(90)90358-O.
URL: http://www.sciencedirect.com/science/article/pii/0012365X90903580 (visited on 06/26/2019).

[Cha10] Timothy M. Chan. "A Dynamic Data Structure for 3-D Convex Hulls and 2-D Nearest Neighbor Queries".
In: *J. ACM* 57.3 (Mar. 2010), 16:1–16:15. ISSN: 0004-5411.
DOI: 10.1145/1706591.1706596.
URL: http://doi.acm.org/10.1145/1706591.1706596.

[CMS95] Jon Christensen, Joe Marks, and Stuart Shieber. "An Empirical Study of Algorithms for Point-feature Label Placement".
In: *ACM Trans. Graph.* 14.3 (July 1995), pp. 203–232.
ISSN: 0730-0301. DOI: 10.1145/212332.212334.
URL: http://doi.acm.org/10.1145/212332.212334 (visited on 05/07/2019).

[Con17a] OpenLayers Contributors. *OpenLayers Project Page*. [Online; accessed 10-February-2018]. 2017. URL: http://openlayers.org/.

[Con17b] OpenStreetMap Contributors. *Map Icons/Proposed Icons — OpenStreetMap Wiki*. [Online; accessed 4-February-2018]. 2017. URL: http://wiki.openstreetmap.org/w/index.php?title=Map_Icons/Proposed_Icons&oldid=1463667.

[Con17c] OpenStreetMap Contributors. *OpenStreetMap Project*. [Online; accessed 26-June-2019]. 2017. URL: https://www.openstreetmap.org.

[Cro85] Robert G. Cromley. "An LP relaxation procedure for annotating point features using interactive graphics". In: *In AUTO-CARTO 7, Proceedings, Digital Representations of Spatial Knowledge*. 1985, pp. 127–132.

[DF92] Jeffrey S. Doerschler and Herbert Freeman. "A Rule-based System for Dense-map Name Placement". In: *Commun. ACM* 35.1 (Jan. 1992), pp. 68–79. ISSN: 0001-0782. DOI: 10.1145/129617.129620. URL: http://doi.acm.org/10.1145/129617.129620 (visited on 10/25/2019).

[Dij+02] Steven Van Dijk et al. "Towards an evaluation of quality for names placement methods". In: *International Journal of Geographical Information Science* 16.7 (Nov. 1, 2002), pp. 641–661. ISSN: 1365-8816. DOI: 10.1080/13658810210138742. URL: https://doi.org/10.1080/13658810210138742 (visited on 01/14/2019).

[DP73] David H Douglas and Thomas K Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), pp. 112–122.

[DZ04] Tamal K. Dey and Wulue Zhao. "Approximating the Medial Axis from the Voronoi Diagram with a ConvergenceGuarantee". In: *Algorithmica* 38.1 (Jan. 1, 2004), pp. 179–200. ISSN: 1432-0541. DOI: 10.1007/s00453-003-1049-y. URL: https://doi.org/10.1007/s00453-003-1049-y (visited on 04/10/2019).

[FA87] Herbert Freeman and John Ahn.
"On the problem of placing names in a geographic map".
In: *Int. J. Pattern Recogn. Artif. Intell.* 1.1 (Apr. 1987), pp. 121–140.
ISSN: 0218-0014. DOI: 10.1142/S0218001487000096.
URL: http://dx.doi.org/10.1142/S0218001487000096 (visited on 05/07/2019).

[FE93] Douglas M. Flewelling and Max J. Egenhofer.
"Formalizing importance: parameters for settlement selection in a geographic database". In: *In Proc. Auto-Carto*. 1993, pp. 167–175.

[FKS16] Stefan Funke, Filip Krumpe, and Sabine Storandt.
"Crushing Disks Efficiently". In: *Combinatorial Algorithms*.
International Workshop on Combinatorial Algorithms.
Lecture Notes in Computer Science. Springer, Cham, Aug. 17, 2016, pp. 43–54. DOI: 10.1007/978-3-319-44543-4_4.
URL: https://link.springer.com/chapter/10.1007/978-3-319-44543-4_4 (visited on 01/04/2018).

[Fre05] Herbert Freeman. "Automated cartographic text placement".
In: *Pattern Recognition Letters*. In Memoriam: Azriel Rosenfeld 26.3 (Feb. 1, 2005), pp. 287–297. ISSN: 0167-8655.
DOI: 10.1016/j.patrec.2004.10.023.
URL: http://www.sciencedirect.com/science/article/pii/S0167865504003381 (visited on 07/06/2017).

[Fre07] Herbert Freeman. "On the problem of placing names on a map".
In: *Journal of Visual Languages & Computing*. In honour of Stefano Levialdi 18.5 (Oct. 1, 2007), pp. 458–469. ISSN: 1045-926X.
DOI: 10.1016/j.jvlc.2007.08.006.
URL: http://www.sciencedirect.com/science/article/pii/S1045926X07000535 (visited on 07/26/2017).

[Fre85] Herbert Freeman. "The automatic labeling of geografic maps - a problem in computer aestetics".
In: *Proceedings Graphics Interface* 84 (May 1985), p. 9.

[Fre86] Herbert Freeman.
A state-of-the-art assessment of automatic name placement.
BATTELLE MEMORIAL INST COLUMBUS OH COLUMBUS LABS, Aug. 1986.
URL: https://apps.dtic.mil/docs/citations/ADA192785 (visited on 05/07/2019).

[Fre95] H Freeman. "On the automated labeling of maps".
In: *Shape, Structure and Pattern Recognition*.
World Scientific, Singapore, 1995, pp. 432–42.

[FW91] Michael Formann and Frank Wagner.
"A Packing Problem with Applications to Lettering of Maps".
In: *Proceedings of the Seventh Annual Symposium on Computational Geometry*. SCG '91.
event-place: North Conway, New Hampshire, USA.
New York, NY, USA: ACM, 1991, pp. 281–288.
ISBN: 978-0-89791-426-0. DOI: 10.1145/109648.109680.
URL: http://doi.acm.org/10.1145/109648.109680 (visited on 07/15/2019).

[Gmb18] Geofabrik GmbH. *Geofabrik GmbH*. [Online; accessed 26-June-2019].
2018. URL: https://www.geofabrik.de.

[GNR11] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter.
"Consistent Labeling of Rotating Maps".
In: *arXiv:1104.5634 [cs]* (Apr. 29, 2011). arXiv: 1104.5634.
URL: http://arxiv.org/abs/1104.5634 (visited on 01/04/2018).

[GNR16] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter.
"Evaluation of Labeling Strategies for Rotating Maps".
In: *J. Exp. Algorithmics* 21 (Apr. 2016), 1.4:1–1.4:21.
ISSN: 1084-6654. DOI: 10.1145/2851493. URL:
http://doi.acm.org/10.1145/2851493 (visited on 01/04/2018).

[Gur19] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*.
2019. URL: http://www.gurobi.com.

[Imh62] Eduard Imhof. "Die Anordnung der Namen in der Karte".
In: *International Yearbook of Cartography* 2 (1962), pp. 93–129.

[Imh75] Eduard Imhof. "Positioning names on maps".
In: *The American Cartographer* 2.2 (Jan. 1, 1975), pp. 128–144.
ISSN: 0094-1689. DOI: 10.1559/152304075784313304. URL: http://www.tandfonline.com/doi/abs/10.1559/152304075784313304 (visited on 07/06/2017).

[inp19] inphos42. *inphos42/osmpbf*. [Online; accessed 26-June-2019].
June 10, 2019. URL: https://github.com/inphos42/osmpbf (visited on 10/11/2019).

[KL78] G.A. Kabatiansky and V.I. Levenshtein.
"Bounds for packings on a sphere and in space,"
in: *Problemy Peredachi Informatsii* 17.1 (1978), pp. 3–25.

[KM20] Filip Krumpe and Thomas Mendel.
Computing Curved Area Labels in Near-Real Time. 2020.
arXiv: 2001.02938 [cs.HC].

[KOS97] Marc Van Kreveld, Rene Van Oostrum, and Jack Snoeyink.
"Efficient settlement selection for interactive display". In: *In Proc. Auto-Carto 13: ACSM/ASPRS Annual Convention Technical Papers*. 1997, pp. 287–296.

[Kre+00] Marc van Kreveld et al.
Computational geometry algorithms and applications. Springer, 2000.
ISBN: 978-3-540-65620-3. URL: http://125.234.102.146:8080/dspace/handle/DNULIB_52011/1394 (visited on 11/13/2019).

[Kru18] Filip Krumpe.
"Labeling Points of Interest in Dynamic Maps using Disk Labels".
In: *10th International Conference on Geographic Information Science (GIScience 2018)*.
Ed. by Stephan Winter, Amy Griffin, and Monika Sester. Vol. 114.
Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 8:1–8:14. ISBN: 978-3-95977-083-5.
DOI: 10.4230/LIPIcs.GISCIENCE.2018.8.
URL: http://drops.dagstuhl.de/opus/volltexte/2018/9336 (visited on 11/30/2018).

[Lee82] D. T. Lee. "Medial Axis Transformation of a Planar Shape".
In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4.4 (July 1982), pp. 363–369. ISSN: 0162-8828.
DOI: 10.1109/TPAMI.1982.4767267.

[LP86] G.E. Langran and T. K. Poiker.
"Integration of Name Selection and Name Placement".
In: *Proceedings, Second International Symposium on Spatial Data Handling, 1986* (1986), pp. 50–64.

[Mat01] Barrault Mathieu.
"A methodology for placement and evaluation of area map labels".
In: *Computers, Environment and Urban Systems*. GISRUK 2000 25.1 (Jan. 1, 2001), pp. 33–52. ISSN: 0198-9715.
DOI: 10.1016/S0198-9715(00)00039-9.
URL: http://www.sciencedirect.com/science/article/pii/S0198971500000399 (visited on 07/07/2017).

[Men18] Natalia Mendel. "Dynamische Beschriftung von Gebietshierarchien - Entwicklung und Implementierung der Beschriftung von hierarchischen Gebietsunterteilungen".
Bachelorthesis. Stuttgart: University of Stuttgart, Oct. 1, 2018. 32 pp.

[Men20] Thomas Mendel.
"Improved Algorithms for Map Rendering and Route Planning".
PhD thesis. University of Stuttgart, 2020. unpublished thesis.

[MP10] David M. Mount and Eunhui Park.
"A Dynamic Data Structure for Approximate Range Searching".
In: *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*. SoCG '10.
Snowbird, Utah, USA: ACM, 2010, pp. 247–256.
ISBN: 978-1-4503-0016-2. DOI: 10.1145/1810959.1811002.
URL: http://doi.acm.org/10.1145/1810959.1811002.

[MS00] Michael McAllister and Jack Snoeyink.
"Medial Axis Generalization of River Networks". In: *Cartography and Geographic Information Science* 27.2 (Jan. 1, 2000), pp. 129–138.
ISSN: 1523-0406. DOI: 10.1559/152304000783547966.
URL: http://dx.doi.org/10.1559/152304000783547966 (visited on 07/06/2017).

[MS91] Joe Marks and Stuart Shieber.
The Computational Complexity of Cartographic Label Placement.
1991.

[Pap81] Christos H. Papadimitriou.
"On the Complexity of Integer Programming".
In: *J. ACM* 28.4 (Oct. 1981), pp. 765–768. ISSN: 0004-5411.
DOI: 10.1145/322276.322287.
URL: http://doi.acm.org/10.1145/322276.322287 (visited on 11/13/2019).

[PF96] I. Pinto and H. Freeman.
"The feedback approach to cartographic areal text placement".
In: *Advances in Structural and Syntactical Pattern Recognition*.
Ed. by Petra Perner, Patrick Wang, and Azriel Rosenfeld.
Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 341–350.
ISBN: 978-3-540-70631-1.

[Pro15] The CGAL Project. *CGAL User and Reference Manual*. 4.7.
CGAL Editorial Board, 2015.
URL: http://doc.cgal.org/4.7/Manual/packages.html.

[Sch+13] Nadine Schwartges et al.
"Optimizing Active Ranges for Point Selection in Dynamic Maps".
In: *Proceedings of the 16th ICA Generalisation Workshop (ICA'13)*.
2013.

[Sch89] Michel Schmitt.
"Some examples of algorithms analysis in computational geometry by means of mathematical morphological techniques".
In: *Geometry and Robotics*.
Ed. by J. -D. Boissonnat and J. -P. Laumond.
Lecture Notes in Computer Science.
Springer Berlin Heidelberg, 1989, pp. 225–246.
ISBN: 978-3-540-46748-9.

[Ved94] Aruna Ashtakala Vedula.
Automatic positioning of area feature names on special purpose maps.
1994.

[Wik19a] OpenStreetMap Wiki. *DE:Grenze — OpenStreetMap Wiki*.
[Online; accessed 16-December-2019]. 2019. URL: %5Curl%7Bhttps://wiki.openstreetmap.org/w/index.php?title=DE:Grenze&oldid=1934872%7D.

[Wik19b] OpenStreetMap Wiki.
Tag:boundary=administrative — OpenStreetMap Wiki.
[Online; accessed 16-December-2019]. 2019. URL: %5Curl%7Bhttps://wiki.openstreetmap.org/w/index.php?title=Tag:boundary%3Dadministrative&oldid=1922057%7D.

[Wik19c] Wikipedia Contributors.
Haversine formula — Wikipedia, The Free Encyclopedia.
[Online; accessed 25-June-2019]. 2019.

[Wik19d] Wikipedia Contributors. *World Geodetic System*. In: *Wikipedia*.
Page Version ID: 919452578. Oct. 3, 2019.
URL: https://en.wikipedia.org/w/index.php?title=World_Geodetic_System&oldid=919452578 (visited on 10/30/2019).

[Wol] Alexander Wolff. *The Map-Labeling Bibliography*.
URL: https://i11www.iti.kit.edu/~awolff/map-labeling/bibliography/ (visited on 05/20/2019).

[WS] David P. Williamson and David B. Shmoys.
The Design of Approximation Algorithms. 500 pp.
URL: http://www.designofapproxalgs.com/download.php (visited on 11/21/2017).

[Wyn65] A. D. Wyner. "Capabilities of bounded discrepancy decoding". In: *The Bell System Technical Journal* 44.6 (July 1965), pp. 1061–1122. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1965.tb04170.x.

[Yoe72] P. Yoeli. "The logic of automated map lettering". In: *Cartographic J.* 9.2 (1972), pp. 99–108. URL: https://ci.nii.ac.jp/naid/10026454235/#cit (visited on 05/07/2019).