TU WIEN Informatics

Conformity-Based Area Labeling on Cartographic Maps

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

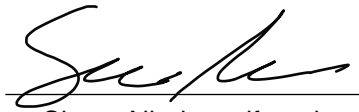
Simon Niederwolfsgruber
Registration Number 12122091

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Assistance: Dipl.-Ing. Thomas Depian

Vienna, November 30, 2024



Simon Niederwolfsgruber | Martin Nöllenburg

Erklärung zur Verfassung der Arbeit

Simon Niederwolfsgruber

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 30. November 2024

Simon Niederwolfsgruber

Kurzfassung

Die Platzierung von Beschriftungen auf Karten ist ein bekanntes Problem in der Kartografie und kann in drei Hauptkategorien eingeteilt werden: Punkt-, Linien- und Flächenbeschriftung. Während die automatisierte Platzierung von Beschriftungen für Punkten und Linien ausgiebig untersucht wurde und es bereits formale Modelle gibt, ist die Forschung zur automatisierten Flächenbeschriftung aufgrund der erhöhten Mess-Komplexität der Beschriftungsqualität noch begrenzt.

Deshalb stellen wir in dieser Arbeit verschiedene Methoden aus der aktuellen Literatur vor und vergleichen diese. Wir konzentrieren uns insbesondere auf Modelle, die Beschriftungen entlang von Kreisbögen platzieren und darauf abzielen, einen hohen Grad an Konformität zu erreichen, das heißt eine hohe Ähnlichkeit einer Beschriftung mit der Fläche, in der sie platziert werden. Im Rahmen dieser Arbeit stellen wir die HeightConstrainedLabeling-Problemvariante sowie eine darauf basierende, in C++ implementierte Pipeline zur Flächenbeschriftung vor. Diese Pipeline zielt darauf ab, einige der Einschränkungen der aktuellen Literatur zu beheben. Im Detail erweitern wir eine etablierte Qualitätsfunktion, indem wir zusätzliche Metriken einbeziehen, und verbessern anschließend die Pipeline durch Clustering und lokale Suche. Zudem werden wir auch die Ergebnisse dieser Implementierung vorstellen, Einschränkungen aufzeigen und mögliche Verbesserungen diskutieren.

Abstract

Label placement on maps is a well-known problem in cartography and can be categorized into three major categories: point-, line-, and area labeling. While automated point- and line labeling have been extensively studied and have established formal models, research on automated area labeling remains limited due to the increased complexity in measuring labeling quality.

To address this, we present and compare in this thesis, different state of the art approaches. We specifically focus on models that place labels along circular arcs within the boundary polygon and aim to achieve a high degree of conformity, i.e., high similarity of a label with the area it is placed in. As part of this thesis, we propose the HeightConstrained-Labeling problem variant, along with an area labeling pipeline implemented in C++. This pipeline aims to address some of the limitations found in the current literature. In detail, we enhance an established quality function by incorporating additional metrics, then further improve the pipeline through clustering and local search mechanisms. We will also present the results of the implemented pipeline, identify its limitations and discuss potential improvements.

Contents

CHAPTER 1

Introduction

Cartographic maps are more important and information-rich than ever and serve as essential tools for everything from navigating public transit to finding restaurants or cafés. As their complexity increases, effective label placement is crucial to preserve readability and clarity. This is achieved through the process known as *map labeling*, a key aspect of map design. Labeling by hand is a labor-intensive task and solutions for automated map labeling have therefore become increasingly important.

Figure 1.1: Example of a curved text label from Google Maps. A possible circular arcs is drawn in red.

Map labeling can be divided into three major categories: point-, line-, and area labels. Especially the labeling of areas, such as administrative regions (countries, states, ...), lakes, etc., poses unique challenges. While formal models for labeling point and line features are well-established, labeling areas is inherently more challenging. This is because measuring quality in area labeling is more complex, as the criteria involved are not straightforward. One important measure of quality is *conformity*, which involves placing labels that accurately represent the area while effectively utilizing the available

space. To enhance conformity, rather than placing labels on horizontal lines, this thesis explores polygon label placement along curves, primarily circular arcs, as can be seen in Figure 1.1.

Previous work by Barrault [Bar01] and Krumpe [Kru20a] introduced methods for the labeling of polygons along circular arcs. In this thesis, we will present, compare and discuss their approaches in detail. We will also highlight their weaknesses and propose solutions in the HeightConstrainedLabeling variant we implemented in C++. Unlike Barrault or Krumpe, the variant allows specifying a label height h that must be respected when placing the label.

1.1 Overview and Structure

First, to get a better understanding of the current literature, we will take a look at the relevant related work in Chapter 2. Then, in Chapter 3, we will present the important area labeling metrics and introduce the HeightConstrainedLabeling variant. Additionally, the necessary labeling background will be discussed.

Our approach to solve the problem follows a *pipeline* structure that consists of two primary phases:

1. The **Position Generation** phase, discussed in Chapter 4, where we generate a set of viable candidate positions based on the *straight skeleton* of a polygon.
2. The **Position Selection** phase, discussed in Chapter 5, where we identify suited candidates by applying quality metrics. This phase also includes pre- and post-processing steps to refine the selection process.

In Chapter 6, we will present and discuss the findings from our C++ implementation of the HeightConstrainedLabeling variant. Finally, we will give a conclusion in Chapter 7.

CHAPTER 2

Related Work

Automatic cartographic label placement has been a widely studied area for a while now. Although the first map labeling metrics were already proposed by Imhof in the 60s (republished in English [Imh75]) and Yoeli [Yoe72], they laid the groundwork for future work and are still widely used today.

In the following years, several attempts to automate label placement have been made. However, the complexity of the problem was still largely unexplored. Marks and Shieber [MS91] were, to be best of our knowledge, the first to show that cartographic label placement for point-features is NP-complete. This was realized using a reduction from the PLANAR 3-SAT problem.

Ahn and Freeman [FA87] proposed the labeling system AUTONAP which also included area-feature labeling. They placed labels along the straight skeleton of the polygon boundary. Doerschler and Freeman [DF92] developed a rule-based algorithm for placing labels on point-, line-, and area-features, also utilizing the straight skeleton. Pinto and Freeman [PF96] proposed a different approach, that does not consider conflicts with other labels. They argued that the placement of area-labels should be carried out first, since it is such a demanding task and the degree of freedom for area-features is greater than for point- or line-features. Edmondson et al. [ECMS96] presented a labeling algorithm for point-, line-, and area-features. Their algorithm also considers the overlap of labels with important map features and not just overlap with other labels. However, the placement of area-features labels was solely based on the centroid of the boundary polygon.

Barrault [Bar01] attempted to place area labels along circular arcs and introduced a new quality measure for area-feature label placement, *perceived coverage*. Perceived coverage refers to an area within the boundary that varies based on label length and the distance from the label to the boundary. Intuitively, the more a label is centered and the further it is from the boundary, the greater the perceived coverage. His model considered labels with a fixed font-size but variable letter- and word-spacing. The presented algorithm first

generates candidate arcs along the straight skeleton of the polygon. Using the perceived coverage metric, the best arc is then selected for placement.

Krumpe's [Kru20a] work is closely related and builds on Barrault's work but aimed to maximize the font-size while maintaining a constant letter- and word-spacing. This approach also accommodated holes within polygons. The algorithm first approximated the straight skeleton and then computed the *clearance* for each edge, defined as the shortest distance to the polygon boundary. The introduced algorithm RALF generated arcs along the edges with the highest clearance to the boundary. They argued that this approach overcomes Barrault's issue of generating many similar arcs while also improving runtime associated with calculation of the perceived coverage.

Van Dijk et al. [DKSW02] classified important label placement rules and introduces a new quality function that includes the *aesthetics*, the *visibility* and the *label-feature association* metrics. Dörschlag et al. [DPP03] presented an algorithm for area label placement, that allows for placement of not only text, but also for example diagrams. Their approach does not stretch text to cover most of a polygon. They rather try to find an exact coordinate that well-represents the bounding polygon but do not account for polygon holes. Similarly, Wu et al. [WDZL16] used a grid approach to assign parcel numbers within areas.

Lu et al. [LDLD19] introduced a novel quality-function and labeling algorithm for labeling of point-, line- and area-features, that uses a hybrid approach consisting of differential evolution and genetic algorithms. For area-features, they first find a reference line using the straight skeleton. For the final placement, they use their methodology for line-feature labeling.

In a recent paper, Oucheikh and Harrie [OH24] explored the use of deep learning in automatic label placement. They concluded that deep networks can supply good label placement according to readability metrics, but there are still improvements possible in terms of readability and association.

CHAPTER 3

Preliminaries

In the following sections, we will present the important metrics used in area labeling and introduce the HeightConstrainedLabeling problem variant, we will implement. Additionally, we will discuss the necessary background for the labeling pipeline, which we will cover in the next chapters.

3.1 Defining Important Metrics

In the 60s and 70s, Imhof [Imh75] and Yoeli [Yoe72] introduced label-placement rules that are still widely used today. These rules state for example that an area must only be labeled once, labels should be placed on circular arcs or horizontal lines and should not touch the boundaries but be spread across the area to cover most of the feature.

Additionally, Barrault [Bar01] defined six criteria that are likely to influence label quality (for examples see Figure 3.1):

Longitudinal extent The longitudinal extent is defined as the extent along the circular arc (left and right), which should be maximized.

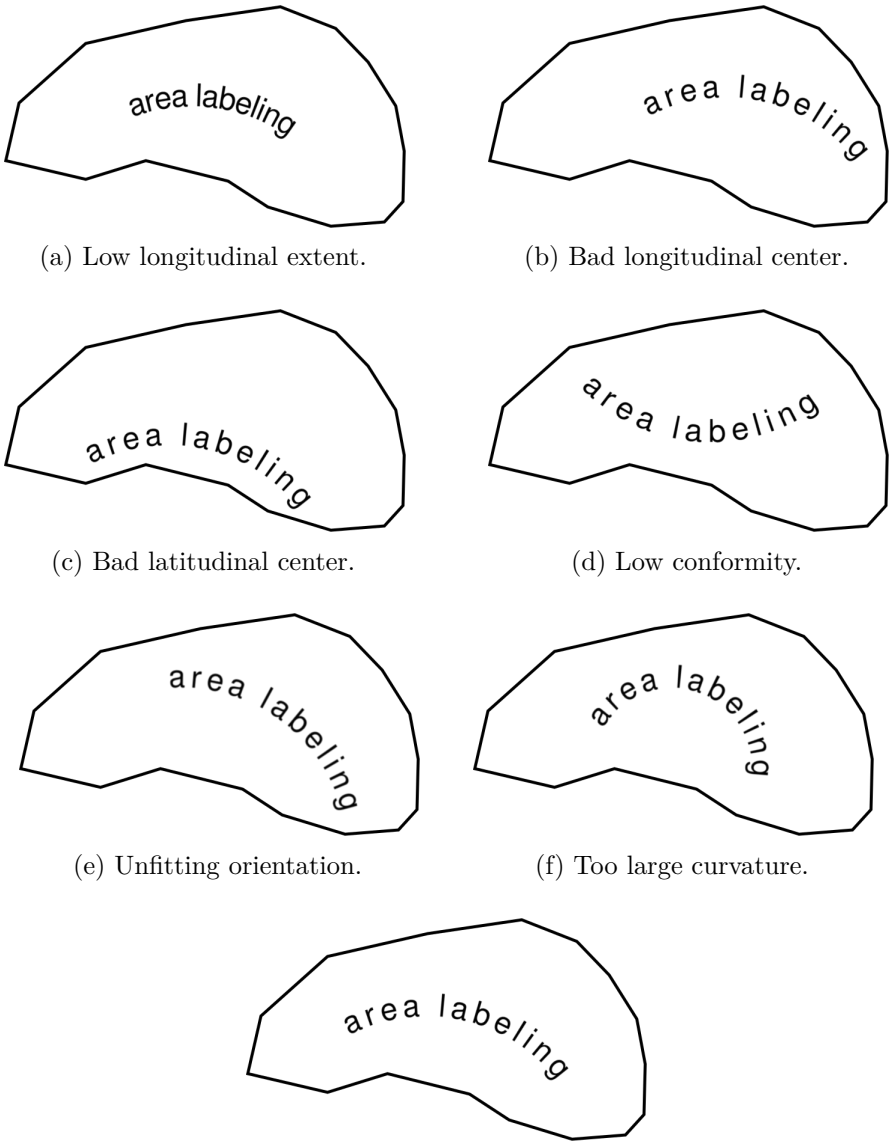
Longitudinal center The label should be centered in the polygon in the longitudinal dimension.

Latitudinal center The label should be centered in the polygon in the latitudinal/vertical dimension

Conformity The shape of the label should have maximal similarity with the area. By looking at the labeled area, it should be clear that the label belongs to the labeled area.

Orientation To improve readability, horizontal labels are generally preferred.

Curvature Circular arcs with larger radii are generally preferred in map labeling, since they offer greater readability.



(a) Low longitudinal extent.

(b) Bad longitudinal center.

(c) Bad latitudinal center.

(d) Low conformity.

(e) Unfitting orientation.

(f) Too large curvature.

(g) Adequate placement.

Figure 3.1: Label placement examples.

3.2 Model

In the general formulation of the area labeling problem, we are given an area to be labeled, along with a text string. The objective is to determine an optimal label placement that

can be contained entirely within the area's boundary.

In this thesis, we will consider the HeightConstrainedLabeling problem variant, we will define in the following.

The input consists of a non-self-intersecting polygon $P \subset \mathbb{R}^2$ and a label height, i.e., font-size h. The output is a placement of a box bent around a circular arc, represented as the tuple $(x, y, R_l, R_u, \beta, \Delta\beta)$ that is visible in Figure 3.2.

We will denote (x, y) as the center of the circular arc with R_l as the lower radius and R_u as the upper radius of the box, we aim to place. If $h = 0$, it follows $R_l = R_u$. Additionally, the placement is characterized by the starting angle $\beta \in [0, 2\pi]$ relative to the positive x-axis and angular extent $\Delta\beta \in (0, 2\pi]$.

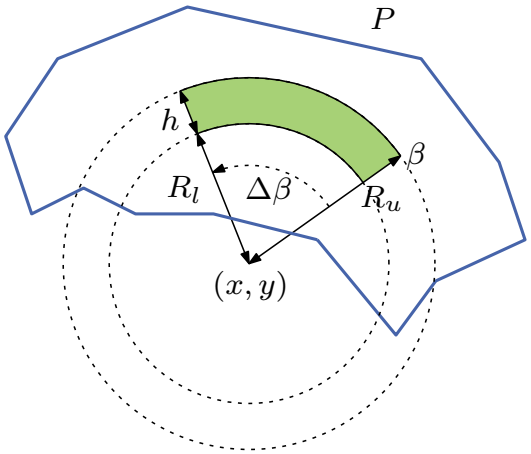


Figure 3.2: HeightConstrainedLabeling model.

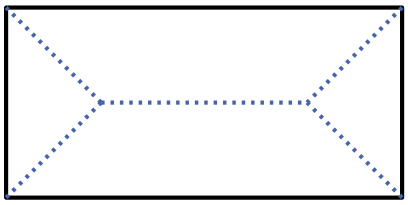
3.3 (Straight) Skeleton and Medial Axis

The *medial axis* or *skeleton* is a line representation of a polygon and is particularly useful, as it provides a reference for the identification of centered segments within the polygon. In contrast to other representations, such as the centroid, the skeleton aims to preserve the shape of the polygon and offers a greater perception of available space. Thus, it can be used to indicate where a label of high conformity could be placed.

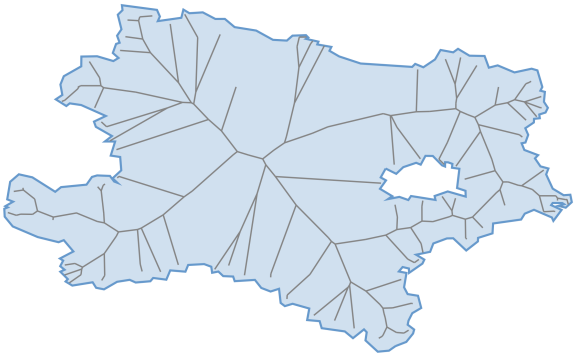
The concept was introduced first by Blum [H67] and Lee [Lee82]. Lee defines the medial axis $M(P)$ of a polygon P as the set of all points $q \in P$ such that at least two points on the polygon's boundary are equidistant and closest to q. An intuitive approach to understanding this structure is by propagating the polygon lines equally inward, parallel to themselves. The points at which the lines intersect create the structure of the medial axis.

The *straight skeleton* limits this representation to only straight lines and is therefore better suited for computer representation. It can be approximated using the *Delaunay*

triangulation, which we will cover next. Examples for the straight skeleton are given in Figure 3.3.



(a) Straight skeleton of a rectangle.



(b) Straight (pruned) skeleton of Lower Austria.

Figure 3.3: Examples for the straight skeleton.

3.4 Delaunay Triangulation and Voronoi Diagram

The *Delaunay triangulation* is a triangulation of a set of points into triangles, with the points representing triangle vertices. More formally, let S be a set of n points in $\mathbb{R}^2$. Now, the Delaunay triangulation is formed by connecting with a line segment any two points $p, q \in S$ for which a circle C exists that passes through p and q and does not contain any other point of S in its interior [DBCVKO08]. The elementary property of

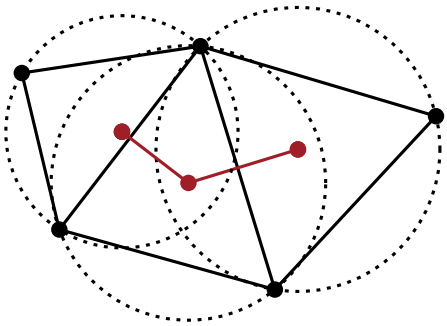


Figure 3.4: Delaunay triangulation of 5 points (black), with the Voronoi diagram and straight skeleton shown in red.

this structure is that the circumcircles of the calculated triangles do not contain any other points from the point set. The centers of the triangle circumcircles form the dual of the Delaunay triangulation, known as the *Voronoi diagram* from which the straight skeleton can be approximated. Given n points, the Delaunay triangulation can be computed in $O(n \log n)$ time. An example of a Delaunay triangulation of a point-set (black) is given in Figure 3.4. The circumcircle centers, shown in red, form the Voronoi diagram. To

approximate the straight skeleton from the triangulation, the circumcircle centers of adjacent triangles, i.e., triangles that share an edge, are connected. Any edges that are not completely contained within the convex hull of P are discarded. The result is the approximated (pruned) straight skeleton.

3.5 Clearance

Krumpe [Kru20a] defines (edge-)clearance for the straight skeleton as the approximated lowest distance of a skeleton edge to the polygon's boundary. The clearance allows us to identify segments within the straight skeleton that best represent the polygon, i.e., offer a high degree of conformity. These segments can then be used as a guide for label placement.

In the previous Section 3.4, we have established that an edge in the straight skeleton is formed by connecting the centers of adjacent triangle circumcircles in the Delaunay triangulation. The shared edge of adjacent triangles is called the *Delaunay edge*, which is not part of the skeleton. Using the properties of the Delaunay triangulation, we can now approximate the clearance using Krumpe's method:

For any given skeleton edge, there are two cases (an example is given in Figure 3.5).

The circumcircle centers are on different sides of the Delaunay edge: In this case, the clearance is defined as half the length of the Delaunay edge.

The circumcircle centers are on the same side of the Delaunay edge: For this case, the clearance is defined as the minimum of the radii of the two corresponding circumcircles.

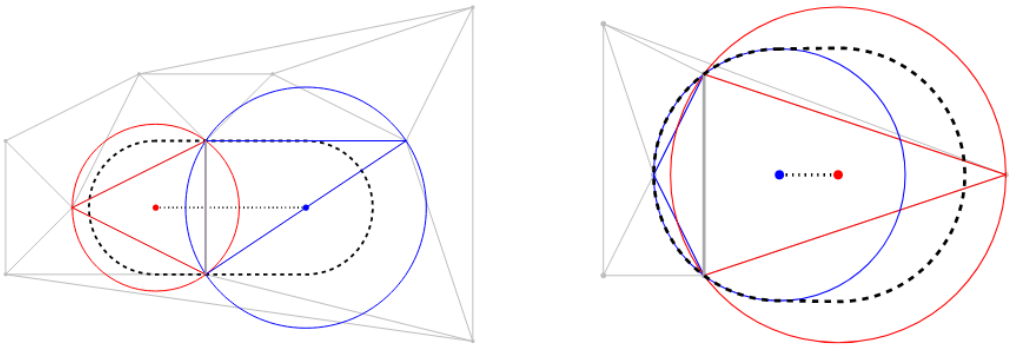


Figure 3.5: The clearance of a skeleton edge if the centers are on different sides of the Delaunay edge (left) or on the same side (right) of the Delaunay edge. The shown graphic is adapted from Krumpe [Kru20a].

CHAPTER 4

Position Generation

The generation of a rich and diverse candidate set is essential for the placement of suitable labels. Some approaches in the literature focus on singular points, e.g., the centroid of the polygon [DPP03, WDZL16] to derive placements. While these methods are effective for labels that do not aim to cover the polygon, they are less suitable for conformity-based approaches. The straight skeleton, discussed in Section 3.3, has shown to be a more effective approach for identifying placements with high conformity. In the following sections, we are going to discuss Barrault's [Bar01] and Krumpe's [Kru20a] position generation strategies, both of which utilize the straight skeleton. Furthermore, we will extend the presented position generation methods in Section 4.4, to also include height constraints.

4.1 Barrault's Path Generation

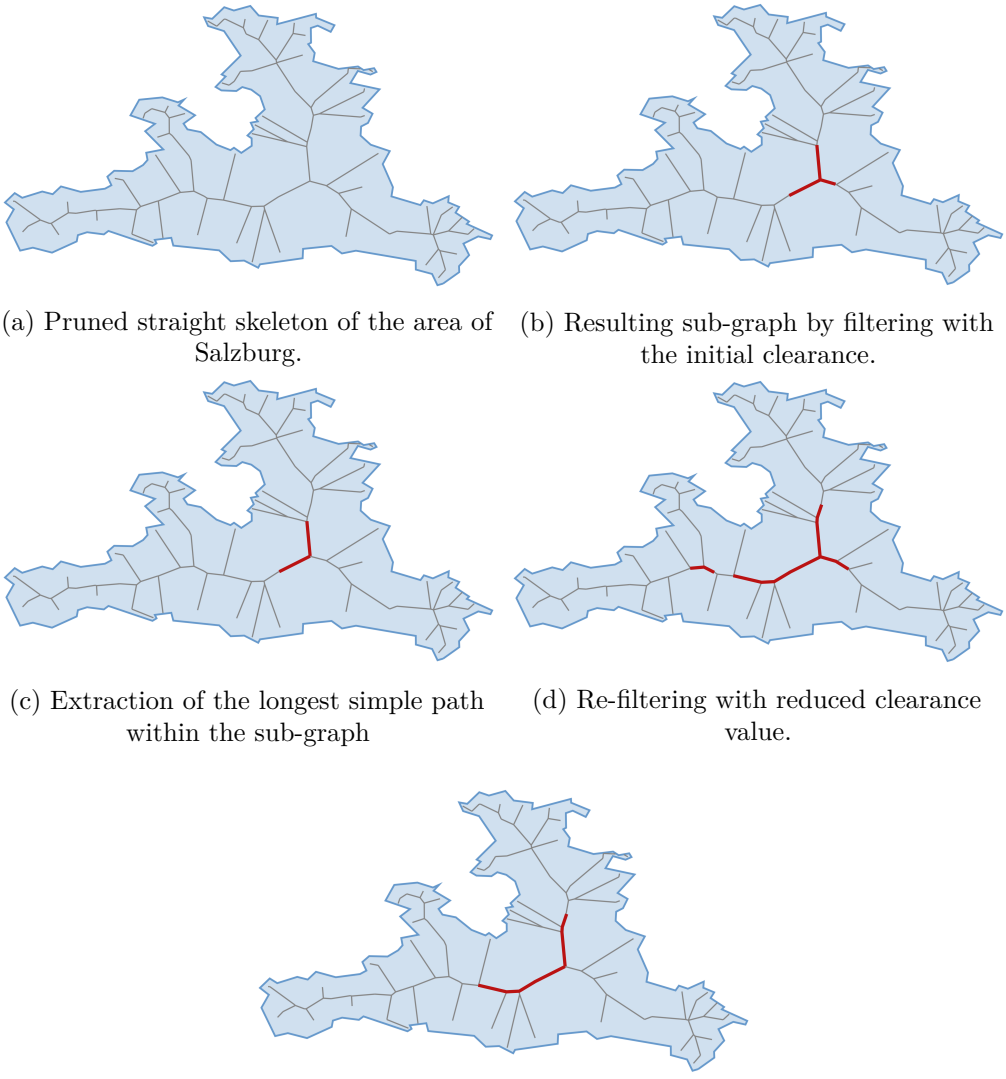
Recall that Barrault's [Bar01] methodology is based on the placement of labels with a fixed label height but a variable letter- and word spacing. Barrault aims to maximize label length when generating the candidate set, but does not directly consider the label height in the position generation.

As a first step in the *generation process*, morphological erosion is applied to the polygon to remove excessive detail. Morphological erosion is an operator that simplifies complex shapes while preserving essential polygon characteristics necessary for accurate labeling. Next, the algorithm approximates the straight skeleton using the Delaunay triangulation, recall Section 3.4. Within the skeleton, Barrault uses Dijkstra's algorithm for each leaf node to retain the $k = 50$ longest *simple paths*[1]. Given a skeleton with $|E|$ edges and $|V|$ vertices, we can compute the set of paths in $O(|V| \cdot |E| \log |V|)$ time. Using the

[1] A simple path is a path, that does not have any repeated vertices.

method we will describe in Section 4.3.1, we fit circular arcs along these paths to yield the candidate positions.

4.2 Krumpe's Path Generation



(a) Pruned straight skeleton of the area of Salzburg.

(b) Resulting sub-graph by filtering with the initial clearance.

(c) Extraction of the longest simple path within the sub-graph

(d) Re-filtering with reduced clearance value.

(e) Extraction of the longest simple path within the new sub-graph.

Figure 4.1: Krumpe's clearance filtering algorithm for the straight skeleton.

Contrary to Barrault [Bar01], Krumpe [Kru20a] sets a fixed aspect ratio a for the label and then maximizes its scale. While Krumpe also uses the straight skeleton for the generation of the candidate set, he follows a more elaborate approach that aims to remedy

the shortcomings of Barrault's method we will discuss in Chapter 6. Rather than solely focusing on the longest simple paths within the skeleton, his methodology aims to find skeleton paths with a high clearance, i.e., the lowest distance of a skeleton edge to the polygon's boundary. Intuitively, the objective is to find paths that allow for a great use of the vertical space. For a more detailed explanation of the clearance value and its calculation, the reader is referred to Section 3.5.

A visualization of Krumpe's algorithm is presented in Figure 4.1. In detail, in the first step, a large clearance value c is selected, e.g., the maximum clearance value within the skeleton. The skeleton is then filtered and edges with a lower clearance value are removed. The result is a sub-graph of the straight skeleton. The longest simple paths for each connected component in the sub-graph are then calculated using Dijkstra's algorithm. We define the minimal length of a pathh as $l_{min} = \frac{2 \cdot c}{a}$. If the length of the path is equal or greater than l_{min}, the path is retained. The clearance c is then reduced by $\sqrt{2}$. The reason behind this is that if the potential height of the label is reduced by $\sqrt{2}$, then its area is halved. This procedure is repeated until $k = 30$ paths are found.

For a skeleton graph with $|E|$ edges and $|V|$ vertices, the filtering can be performed on $O(|E|)$ time. Then need to perform Dijkstra's algorithm on each connected component $g \in G$. The total computational cost across all connected components can be bounded by:

$$\sum_{g \in G} (|E_g| \log |V_g|) \leq |E| \log |V|$$

where $|E_g|$ and $|V_g|$ denote the number of edges and vertices in each connected component. Thus, the overall time complexity becomes $O(i \cdot |E| \log |V|)$, where i represents the number of iterations needed. Since not every path is retained, we might need more than k filtering steps. A reasonable upper bound for i is $\log_{\sqrt{2}}(\frac{c_{max}}{c_{min}})$, as the clearance decreases by a factor of $\sqrt{2}$ in each iteration. Here, c_{max} and c_{min} denote the maximum and minimum clearance value within the skeleton, respectively.

Similarly to Barrault, Krumpe approximates circles along these paths using the method we will describe in Section 4.3.1 to yield the candidate positions.

4.3 Path Approximation

The algorithms presented in the previous Sections 4.1 and 4.2 only give us simple paths within the straight skeleton (effectively polylines). However, our goal is to place labels on a curve. Imhof [Imh75] recommends the use of a single circular arc and to avoid more complex curves due to readability and map simplicity concerns. He makes an exception however, for long labels, allowing the use of doubly curved lines and states that they can have a very elegant effect. Barrault [Bar01] and Krumpe [Kru20a] limit their approach to the use of a single circular arc. In addition to enhancing readability and map simplicity, this choice significantly simplifies the *selection process* we will discuss in the next chapter.

In the following sections, we will discuss a path approximation method based on circles, but will also consider a different approach using Bézier curves.

4.3.1 Circle Approximation

Given a sequence of path coordinates $(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)$, the objective is to approximate it via a circle with radius R and center (x, y). For this, we use an approach from Thomas [TC89] that generates a circle such that the following error term is minimized.

In detail, the error term is defined as the difference between the constant area $R^2\pi$ and the circle at (x, y) with radius $\sqrt{(x_i - x)^2 + (y_i - y)^2}$. Summing up the squares of errors we have

$$e(R, x, y) = \sum_{i=1}^{n} [\pi R^2 - \pi\{(x_i - x)^2 + (y_i - y)^2\}]^2.$$

We then differentiate with respect to x, y and R. By using a trick and solving for x and y respectively, we can rewrite the results in matrix form.

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

Using the notation

$$\sum_y = \sum_i y_i, \qquad \sum_{x^2} = \sum_i x_i^2, \qquad \sum_{y^2} = \sum_i y_i^2$$

$$\sum_{xy} = \sum_i x_i y_i, \qquad \sum_{x^3} = \sum_i x_i^3, \qquad \sum_{y^3} = \sum_i y_i^3$$

$$\sum_{x^2 y} = \sum_i x_i^2 y_i, \qquad \sum_{xy^2} = \sum_i x_i y_i^2,$$

we get

$$\begin{aligned} a_1 &= 2(\textstyle\sum_x^2 - n\sum_{x^2}), \quad b_1 = 2(\sum_x \sum_y - n\sum_{xy}) \\ a_2 &= 2(\textstyle\sum_x \sum_y - n\sum_{xy}) = b_1, \quad b_2 = 2(\sum_y^2 - n\sum_{y^2}) \\ c_1 &= (\textstyle\sum_{x^2}\sum_x - n\sum_{x^3} + \sum_x\sum_{y^2} - n\sum_{xy^2}) \\ c_2 &= (\textstyle\sum_{x^2}\sum_y - n\sum_{y^3} + \sum_y\sum_{y^2} - n\sum_{x^2 y}). \end{aligned}$$

We will use the values to find the center (x, y) of the approximated circle.

$$\begin{aligned} x &= \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1} \\ y &= \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1} \end{aligned}$$

We then calculate the approximated radius

$$R^2 = \frac{1}{n}(\sum_{x^2} - 2\sum_x x_c + nx_c^2 \sum_{y^2} - 2\sum_y y_c + ny_c^2).$$

To extract a valid circular arc from the circle, we first calculate the intersection points of the circle with the polygon boundary. From the intersection points, we determine the angles θ_j relative to the circle's positive x-axis, with $j = 1, 2, \ldots, k$. This allows us to define angle intervals (θ_j, θ_{j+1}) for $j = 1, 2, \ldots, k-1$, along with the interval (θ_k, θ_1) that connects the last to the first intersection point. From the intervals that lie within the polygon, we retain the one with the largest angular extent as our candidate position and will denote it by the starting angle β and angular extent $\Delta\beta$.

4.3.2 Bézier Curve Approximation

Imhof [Imh75] advises against the use of arbitrary curves to keep maps more simple and readable. However, due to the nature of Bézier curves, they can follow more complex paths if compared to simple circular arcs. This is especially helpful since the generated skeleton paths can have multiple directional changes that circles cannot represent accurately. Since, in this thesis, the primary focus lies on achieving a high degree of conformity the raised concerns will be set aside momentarily to explore how conformity might be improved through the use of Bézier curves for path approximation.

A Bézier curve is an *approximating curve*, i.e., control points do not have to lie on the curve itself. A curve of degree d has $d + 1$ control points. We are going to use the cubic Bézier curve since it allows for a maximum of one inflection point and for us provides the best trade-off between approximation accuracy and simplicity/readability.

The cubic Bézier curve is formed by the four control points P_0, P_1, P_2, P_3 (see Figure 4.2). The first control-point P_0 and last control-point P_3 lie directly on the curve, while P_1 and P_2 do not necessarily.

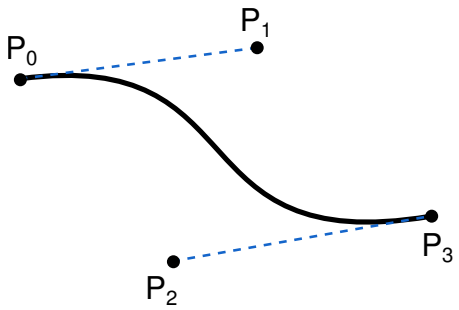


Figure 4.2: Bézier curve example.

Now for a simple approximation of a sequence of path coordinates

$$(x_1, y_1), (x_i, y_i), \ldots, (x_n, y_n)$$

with a Bézier curve, let

$$P_0 = (x_1, y_1), \quad P_3 = (x_n, y_n),$$

P_1 be the point at $\frac{1}{3}$ and P_2 be the point at $\frac{2}{3}$ of the Euclidean length of the path such that all P_i are equally spaced on the path.

4.4 Height Constraints

In this thesis, we aim to solve the HeightConstrainedLabeling problem, but we have not yet directly considered label height in the generation process. Currently, we cannot guarantee that a candidate position, when assigned a height, can be completely contained within the polygon boundary. In this section, we therefore extend the position generation step to include a label height h, such that labels do not extend outside the boundary.

Barrault [Bar01] and Krumpe [Kru20a] base their position generation on a singular circle with radius R. To keep labels from extending beyond the boundary, we are going to use the circle with radius R as a guide only. According to our model, recall Section 3.2, we are going to use two circles instead: one circle that represents the top boundary of the label with radius $R_u = R + \frac{h}{2}$ and one that represents the lower boundary with radius $R_l = R - \frac{h}{2}$.

To extract viable arcs on the circles with R_u and R_l, we first calculate the intersection points of the two circles with the polygon boundary. Next, for each circle, we determine the set of angle intervals that lie within the polygon, similar to Section 4.3.1. However, here, we additionally have to calculate the interval intersections of both circles. We will retain the interval intersection with the largest angular extent as the candidate position and denote by starting angle β and angular extent $\Delta\beta$.

CHAPTER 5

Position Selection

In the second part of the pipeline, the *selection process*, we evaluate and compare candidate arcs according to certain quality metrics. The most suitable is then selected for label placement. Identifying high-quality paths is crucial since their quality can vary significantly.

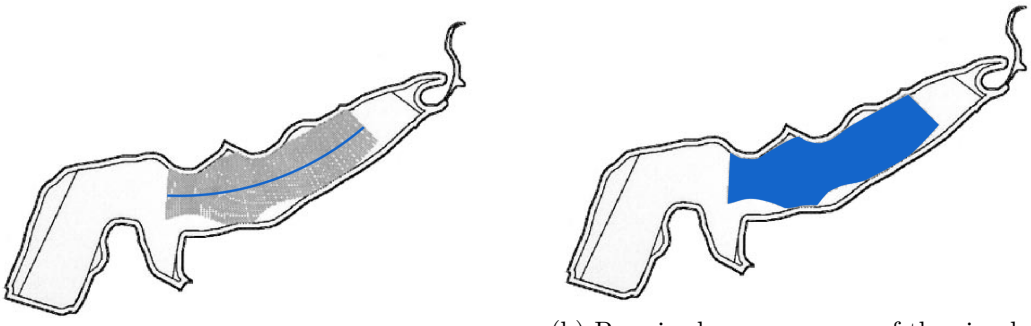
First, in Section 5.1, we will look at Barrault's selection process based on the *perceived coverage* metric. While Krumpe's approach is also very interesting, it is not suited for the HeightConstrainedLabeling variant, since it follows the goal of maximizing the font-size, which is not our goal. Next, in Section 5.2, we will extend the perceived coverage metric by incorporating metrics for *curvature* and *orientation*. Finally, in Section 5.3, we will introduce pre- and post-processing steps to refine the selection process.

5.1 Barrault's Selection Process

Out of the metrics mentioned in Section 3.1, conformity has the most loose definition and is therefore arguably the most difficult to measure. An interesting approach in the literature focusing on conformity is Barrault [Bar01]. He introduced the quality measure known as *perceived coverage* (PC).

The intuition behind this quality measure is presented in Figure 5.1. Consider a circular arc with a height $h = 0$ (Figure 5.1a). We then gradually increase the height equally at the top and bottom, individually for each point on the arc, until either the lower or upper part of the arc touches the boundary. The resulting (shaded) area in Figure 5.1b is the perceived coverage of the circular arc.

However, contrary to HeightConstrainedLabeling, Barrault does not directly consider label height and defines a label placement as the following (a visualization of the terms defined next is given in Figure 5.2). A label placement is characterized by a *support line* SL and a *baseline* B. The support line $SL(x_c, y_c, R, \alpha, \Delta\alpha)$ represents a circular arc



(a) Initial circular arc with height $h = 0$.

(b) Perceived coverage area of the circular arc.

Figure 5.1: Perceived coverage intuition. Graphic adapted from Barrault [Bar01].

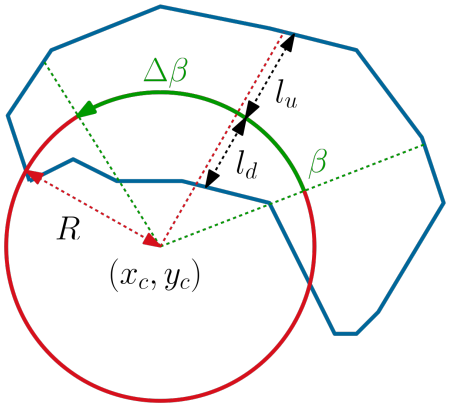


Figure 5.2: Perceived coverage notation.

within P. SL has a radius R and center (x_c, y_c). The support line spans from boundary to boundary within P and is characterized by the starting angle $\alpha \in [0, 2\pi]$ relative to the positive x-axis and angular extent $\Delta\alpha \in (0, 2\pi]$. The baseline represents the final placement of the label, defined along the support line. It is defined as $B(SL, \beta, \Delta\beta)$, with a starting angle $\beta \in [\alpha, \alpha + \Delta\alpha]$ and angular extent $\Delta\beta \in (0, 2\pi]$, such that $\beta + \Delta\beta \leq \alpha + \Delta\alpha$.

We denote $l_d(s)/l_u(s)$ as the lower/upper distance of a point s on the baseline B to the closest lower/upper point of the boundary in the direction described by the vector from the center (x_c, y_c) of the circle to the point s.

Now, for a baseline B, the perceived coverage is the continuous sum of the minimal distances along it (multiplied by 2):

$$PC(B) = 2\int_{\beta}^{\beta+\Delta\beta} pc(s)ds = 2\int_{\beta}^{\beta+\Delta\beta} min(l_d(s), l_u(s))ds$$

This is approximately the shaded area in Figure 5.1. With this measure, we can identify baselines that offer good centering and space in the latitudinal, i.e, "vertical" dimension.

However, for an adequate placement, we need to consider label position and extent in the longitudinal, i.e., "horizontal" dimension as well. Therefore, Barrault extends the metric by considering the longitudinal extent and longitudinal centering of the baseline, recall Section 3.1.

For this, Barrault uses a rule from Cuenin [Cue72] that states that the longitudinal extent e of the baseline should be $\frac{2}{3}$ the length of the support line it is placed on. Additionally, to favor longitudinal centering, a penalty is induced relative to the degree of violation.

$$PC_{SL}(B) = \int_{max(\alpha,\beta-\Delta\beta\cdot(1-e)/2)}^{min(\alpha+\Delta\alpha,\beta+\Delta\beta+\Delta\beta\cdot(1-e)/2)} pc(s)ds \tag{5.1}$$

$$- \int_{\alpha}^{\alpha-min(0,\alpha-\beta-\Delta\beta\cdot(1-e)/2)} pc(s)ds \tag{5.2}$$

$$- \int_{\alpha+\Delta\alpha+min(0,\alpha+\Delta\alpha-(\beta+\Delta\beta+\Delta\beta\cdot(1-e)/2))}^{\alpha+\Delta\alpha} pc(s)ds \tag{5.3}$$

In detail, in Equation (5.1), the perceived coverage for a baseline B along a support line SL is calculated. However, we extend the interval of B on both sides such that the residual extent ($1 - e = \frac{1}{3}$) is shared equally on both sides. Additionally, we do not compute the area beyond the endpoints α and $\alpha + \Delta\alpha$. In Equations (5.2) and (5.3) we penalize arcs being too close to the left and right boundary by removing the area corresponding to the degree of violation.

For the complete metric, we define Q_{cov} on $[0, 1]$, that measures the coverage relative to the polygon area. We denote by S the area of the polygon. To increase the variation between the best candidates, we take the square root of $\frac{PC_{SL}(B)}{S}$.

Thus, the PC for a baseline B can be calculated with the following formula.

$$Q_{cov} = \sqrt{\frac{PC_{SL}(B)}{S}}$$

5.2 Extending Barrault's Quality Function

Barrault's approach focuses solely on coverage while ignoring metrics like curvature and orientation. He argues that these metrics should only intervene in the final selection and are dependent on the cartographer's preference.

In the following, we will extend Barrault's PC metric and define such a quality function that focuses on conformity but also takes curvature and orientation into account.

5.2.1 Curvature

In area labeling, placements on circular arcs with smaller radii are usually avoided, since they are harder to read. Additionally, in terms of conformity, we can often observe that

arcs with larger radii are more effective in representing the more significant parts of a polygon. Therefore, we require a metric to favor arcs with larger radii to ensure that the most suitable arcs are selected.

Similarly to Q_{cov}, we construct the metric

$$Q_{cur} = \sqrt{\frac{R}{R_{max}}}$$

to quantify the relative size of a candidate's radius R compared to the largest radius R_{max} in the candidate set.

5.2.2 Orientation

Vertical text is often hard to read, especially in dense labelings. Therefore, a function is required that favors horizontal placements, while also not completely excluding more vertical options. For this, we aim to find a simple approach using the label placement's center-angle ($\beta + \frac{\Delta\beta}{2}$) as a guide.

The function takes the angle as input and should output values in the interval $[0, 1]$ representing the label's orientation quality. The maximum should be at $\frac{\pi}{2}$ (north) and $\frac{3\pi}{2}$ (south), while the minimum should be 0 (east) and π (west), relative to the positive x-axis. This behavior can be achieved using the simple trigonometric function

$$Q_{ori} = cos^2(\frac{\pi}{2} + (\beta + \frac{\Delta\beta}{2})).$$

5.2.3 Putting it all together

Extending Barrault's proposed PC metric into a more complete quality function yields

$$\begin{aligned} Q = & \, w_{cov} \cdot Q_{cov} \\ & + w_{cur} \cdot Q_{cur} \\ & + w_{ori} \cdot Q_{ori} \end{aligned}$$

Where w_{cov}, w_{cur} and w_{ori} denote the weights of the corresponding metrics and can be chosen individually. With conformity being of greater concern to us than readability, we can confidently say that coverage should be prioritized and given more weight. Curvature plays a smaller, secondary role, while still having some effect on conformity. Orientation is of relevant primarily for readability, rather than conformity and should therefore be given less weight.

5.3 Enhancements

In the following, we will present ways to enhance the position selection step.

Clustering. In map labeling, it is important to consider potential overlap with other features or labels on the map. The problem of finding label placements with the least overlaps was even shown to be NP-complete [MS91]. A diverse set of paths is advantageous since obstructions can be more easily avoided. Initial experiments revealed a high degree of similarity among the candidate paths. To address this, clustering can be applied to group similar paths, ensuring that positions from different clusters are included in the final results. Therefore, as a preprocessing step, we will employ an *agglomerative clustering* algorithm.

The first step is to define a function that determines the similarity of a pair of paths. An intuitive approach is to measure the ratio of overlap between two paths. A high overlap ratio suggests that a significant portion of the paths coincide, indicating that the paths are indeed similar.

More formally, we introduce a similarity metric for two paths S and T, as:

$$Sim(S,T) = \frac{d(S \cap T)}{d(S \cup T)},$$

where $d(S \cap T)$ denotes the Euclidean length of the intersecting segments of S and T, while $d(S \cup T)$ denotes the Euclidean length of the union of their segments.

Now, for the algorithm, each path is initially assigned to its distinct cluster. The next step involves identifying the pair of clusters that demonstrates the highest *average similarity*. For clusters C_1 and C_2, the average similarity is computed as:

$$\frac{1}{|C_1||C_2|} \sum_{S \in C_1} \sum_{T \in C_2} Sim(S,T)$$

We then merge the pair of clusters with the highest average similarity. Finally, we recalculate the average similarity for all remaining clusters to identify and merge the next pair with the highest average similarity. This process is repeated until the amount of clusters is reduced to m.

Local Search. In the following, we will introduce a local search approach that we can use as a post-processing step in order to find local maxima to improve the quality of promising arcs even further.

Let a_0 be an initial arc with center (x, y), radius R, starting angle β and angular extent $\Delta\beta$. Now, our goal is to find neighboring successor arcs that are of higher quality. In detail, we alter the position of the arc by a translation along the vector formed by the center of the circle (x, y) and the mid-point on the arc itself (see Figure 5.3). We can position the neighboring arcs either above, below, or directly on the initial arc and also increase the radius. (Concrete values will be given in Section 6.2.) Thus, the neighborhood $\mathcal{N}$ is defined as the set of arcs generated through this combination of translation and scaling operations. From this neighborhood, the arc with the highest quality is selected, and its neighborhood is subsequently explored. As described in Algorithm 5.1, this iterative process continues until a local maximum or k iterations is reached.

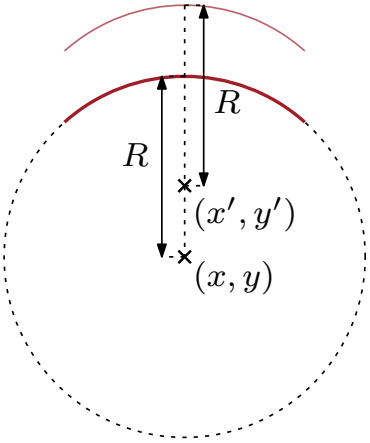


Figure 5.3: Circular arc translation for local search.

Algorithm 5.1: Local search to improve the quality of an arc

Data: Initial arc a_0
Result: Locally optimal arc a^*

$a^* \leftarrow a_0$;
generate neighborhood $\mathcal{N}(a^*)$;
while $\exists a \in \mathcal{N}(a^*)$ *such that* $Q(a) > Q(a^*)$ *and less than* k *iterations* **do**
    $best \leftarrow \arg\max_{a \in \mathcal{N}(a^*)} \mathrm{Q}(a)$;
    update neighborhood $\mathcal{N}(a*)$;
end
return a^*;

CHAPTER 6

Results and Discussion

We implemented the HeightConstrainedLabeling pipeline in C++ utilizing the *Computational Geometry Algorithms Library* (CGAL) [CGA24]. This implementation builds upon Krumpe's open-source project [Kru20b], which among other functionalities includes the skeleton calculation and position generation.

Our implementation follows the introduced pipeline structure and offers a high degree of flexibility. We can specify a polygon P including holes, a label height h and the number of candidates. These parameters will be used for the generation step, where either Barrault's or Krumpe's approach can be selected. Next, in the position selection step, we detect the top placements with our introduced quality function Q where we can freely adjust the weights for perceived coverage, curvature and orientation. Additionally, we can enable or disable the enhancement methods of clustering and local search as needed.

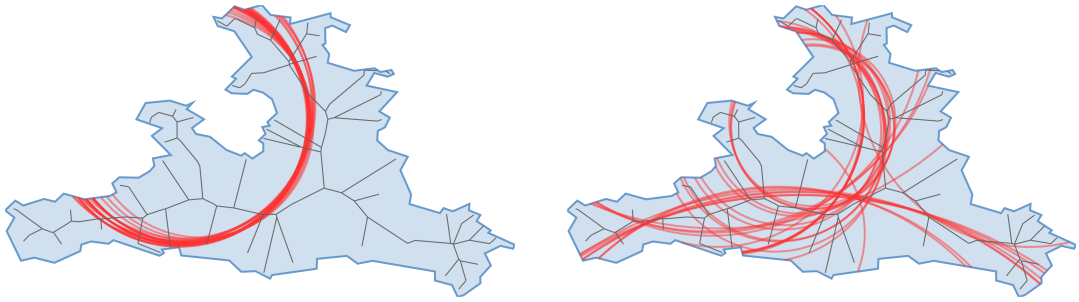
To visualize the results, we utilized Python along with the Matplotlib library. The areal data of Austrian federal states used in our analysis was based on the data from Statistik Austria's Open.data platform [Ope24].

Experiments were carried out on an off-the-shelf laptop with 8 GB of RAM and an older-generation Intel Core 7200U CPU.

The next sections will cover the implementation results of the generation process, followed by the selection process, including its enhancements. Finally, we will briefly look at the runtime we can expect from the algorithm.

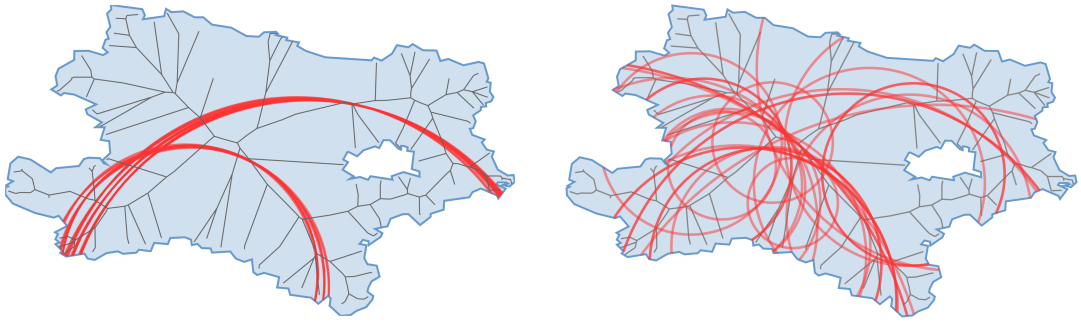
6.1 Position Generation

Barrault vs. Krumpe. In Sections 4.1 and 4.2, we present Barrault's and Krumpe's methods for position generation. For comparison, we present implementation results in



(a) Barrault's position generation (b) Krumpe's position generation

Figure 6.1: Position generation Barrault and Krumpe, Salzburg, $k = 30$ candidates

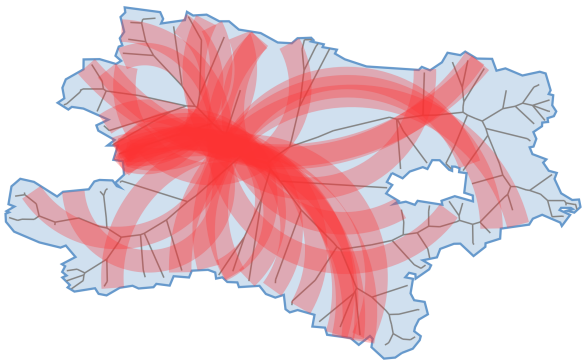


(a) Barrault's position generation (b) Krumpe's position generation

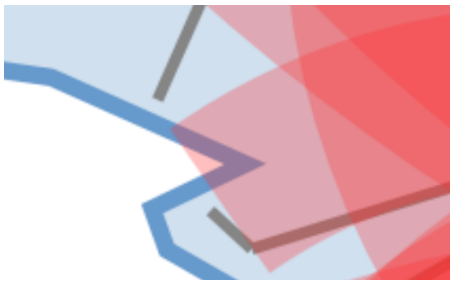
Figure 6.2: Position generation Barrault and Krumpe, Lower Austria, $k = 30$ candidates

Figures 6.1 and 6.2. Comparing Barrault's and Krumpe's methods makes the shortcomings of Barrault's approach evident. In Figure 6.1, with Barrault's method, all arcs lie in the same visual cluster, leaving the lower-right third of the polygon completely unaddressed. This clustering is typical for Barrault's approach, which often produces one to three clusters of arcs and also persists when the number of skeleton vertices is varied. Furthermore, his approach fails to detect horizontal positions at the bottom of the polygon, which are well-suited for label placement. Barrault acknowledged the lack of diversity in his paper, noting that the longest simple paths in the straight skeleton tend to be highly similar. In contrast, Krumpe's method produces a more diverse candidate set and distributes the arcs more evenly across the polygon. In Figure 6.2, Barrault's method can find more adequate placements. However, his candidate set still lacks diversity when compared to Krumpe's.

Krumpe [Kru20a] also critiques that Barrault's method is often unable to generate arcs with larger radii and smaller curvature. He specifically references a legibility metric by Imhof [Imh75], which suggests a minimal angular extent of $\frac{\pi}{3}$. In summary, when compared to Barrault, Krumpe's approach produces a more diverse candidate set, not only in arc location but also in radius size.



(a) Position generation of labels with explicit height, Lower Austria, 30 candidates



(b) Intersection issue with explicit height labels.

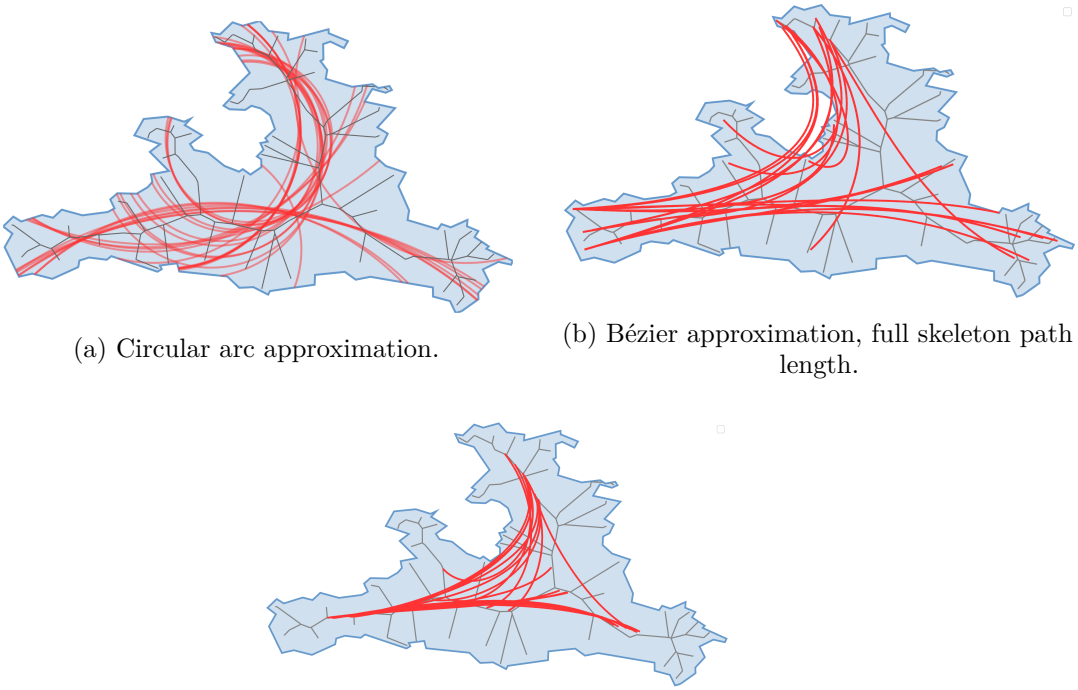
Figure 6.3: Labels with explicit height.

Height Constraints. In Section 4.4, we introduced an algorithm that, given a generated circle and label height, aims to calculate a label position that is completely contained within the polygon. Results are displayed in Figure 6.3a.

There are small flaws with our approach, however. Intersections with the boundary on either end of the arc may not be detected correctly. An example is visible in Figure 6.3b. The reason for this are thin peaks in the boundary that lie exactly between the upper and lower boundary of the label. However, in most real-world areas, this is not a significant concern since the peaks are usually small and we are not going to utilize the full label length in the position selection step. Still, this issue could be remedied in the future by identifying the polygon edges that intersect with the upper and lower label boundary. We would then traverse the edges between the two intersection points and adjust the angular range of the label accordingly. Furthermore, the current method fails to detect sufficiently small holes that are entirely contained within the label. To correct this, we would have to iterate over all the holes within the polygon and verify whether they are contained within the label.

Path Approximation. In Section 4.3, we discussed methods to approximate paths using circles and Bézier curves. We compared the approximation methods directly in Figure 6.4 to make their differences visible. For paths with minimal change in direction, the Bézier approximation appears to be more accurate than for circles and aligns more closely with the skeleton edges. On the other hand, for paths that have larger changes in direction and appear particularly curved, seemingly the opposite is true. The Bézier curves cannot follow the path as well as the circles, and can often not even be contained within the polygon.

Cuenin [Cue72] states that a label should not cover the entire width of the polygon. Instead, the horizontal extent of a label should be $\frac{2}{3}$ of the width of the area. Similarly to Barrault in Section 5.1, we apply this rule and will only use $\frac{2}{3}$ of the length of the



(a) Circular arc approximation.

(b) Bézier approximation, full skeleton path length.

(c) Bézier approximation, $\frac{2}{3}$ length of skeleton path.

Figure 6.4: Approximation of skeleton paths using circular arcs and Bézier curves, Salzburg, with 30 candidates.

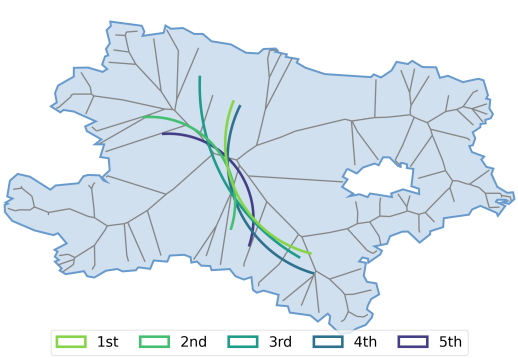
actual path with even spacing on both sides. Now, horizontal curves at the bottom of the polygon yield favorable placements, while the more vertical options still mostly fall short. Even if they are completely contained within the polygon, they still are very close to the boundary and have an uneven curvature.

The reason for this lies in the choice of control points P_1 and P_2. Placing them exactly on the path will not provide optimal results for accurately following a path. By using local search on P_1 and P_2, to minimize the area between the skeleton path and the Bézier curve, the curves could be further improved. This approach does come with its limitations, however. Fitting the curve more closely to the skeleton path will result in a more uneven curvature overall.

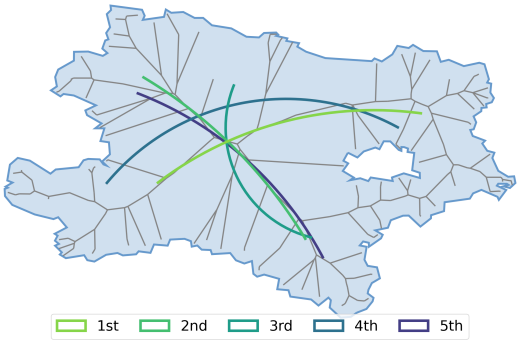
In summary, when path direction changes are minimal, Bézier curves can enhance conformity by closely aligning with the skeleton. However, in the general case, the application of Bézier curves as used here does not substantially improve conformity and is often less effective than using circles. While there exist methods that could achieve closer alignment with the skeleton, they would likely compromise readability due to the uneven curvature.

6.2 Position Selection Results

In this section, we will cover the implementation results of the selection process, while relying on Krumpe's position generation algorithm as a starting point, since it can provide a more diverse candidate set.

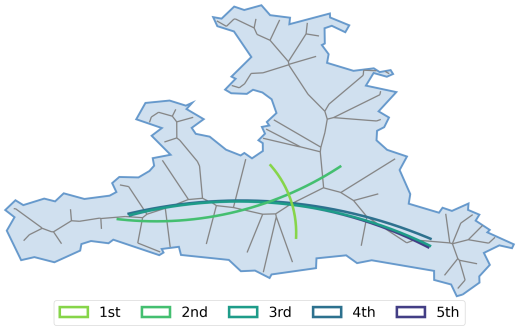


(a) Position selection based on perceived coverage only.

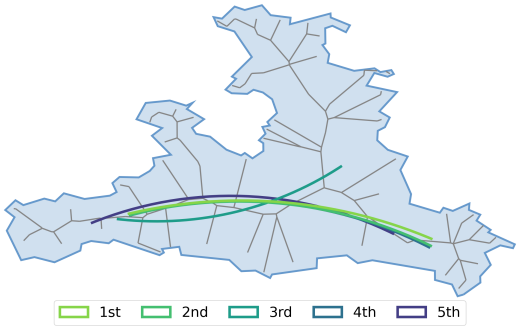


(b) Position selection based on quality function Q with $w_{cov} = 1.0, w_{cur} = 0.25, w_{ori} = 0.1$.

Figure 6.5: Position selection comparison, Lower Austria, top 5 positions.



(a) Position selection based on perceived coverage only.



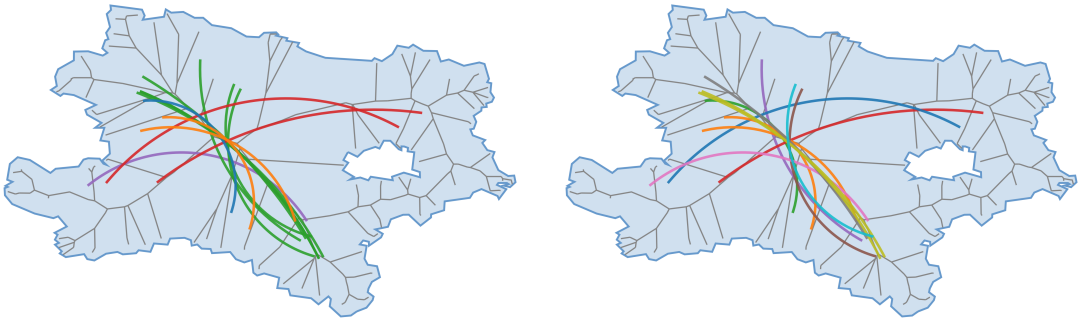
(b) Position selection based on quality function Q with $w_{cov} = 1.0, w_{cur} = 0.25, w_{ori} = 0.1$.

Figure 6.6: Position selection comparison, Salzburg, top 5 positions.

Quality Function Analysis. In Section 5.2.3, we introduced the quality function Q that includes the label's perceived coverage, curvature and orientation. Comparing Q with a selection based solely on perceived coverage reveals significant differences (see Figures 6.5 and 6.6). When position selection is based only on perceived coverage, more vertical arcs with smaller radii are often selected. However, for the best readabilty and aesthetics we want relatively horizontal arcs with larger radii. To address this, we

introduced the quality function Q, which incorporates metrics for both orientation and curvature. This allows Q to identify placements with higher readability and aesthetics in the same candidate set, while still maintaining a strong focus on conformity.

Clustering. We introduced clustering as a post-processing step to achieve more diversity among the top paths. The results of this procedure are shown in Figure 6.7. When comparing Figures 6.7a and 6.7b, we can observe that selecting only 5 clusters is insufficient for this area. For instance, the green and the red cluster could be ideally split into two. Subdividing into 10 clusters, as seen in Figure 6.7b, addresses these issues to some extent. However, in certain cases, it clustered too finely. For example, the cyan and brown clusters could be combined, as could the gray and yellow clusters. Overall, when using $k \approx 30$ candidates, we recommend choosing between 5 and 10 clusters for most areas.



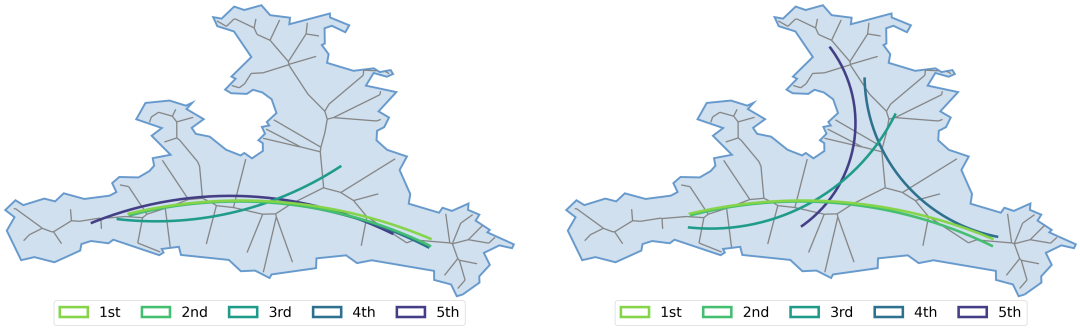
(a) Clustering with $m = 5$ clusters. (b) Clustering with $m = 10$ clusters.

Figure 6.7: Clustering results, Lower Austria.

Selection results with clustering are displayed in Figures 6.8 and 6.9. In Figure 6.8, clustering shows clear improvements in diversity. When clustering is not applied, the selected paths tend to be very similar, leading to a significant redundancy (only arcs at the bottom of the area were selected). This redundancy is significantly reduced after implementing clustering and a set is chosen where all the arcs lie in different parts of the area. However, in the case of Lower Austria in Figure 6.9, there a no clear improvements visible due to the poorer clustering discussed previously. Thus, while clustering is effective, selecting the optimal number of clusters can significantly enhance the results.

Local Search. In Section 5.3, as a post-processing step, we introduced a local search algorithm to improve the quality of promising arcs further. Results are visible in Figures 6.10 and 6.11 using a maximum of 10 iterations. The neighborhood for this search consisted of six arcs, generated by combining two radius increases (5% and 10%) with three translation shifts (-5%, 0% and 5%). For the example in Figure 6.10, we could measure an increase in the quality function Q of around 5%.

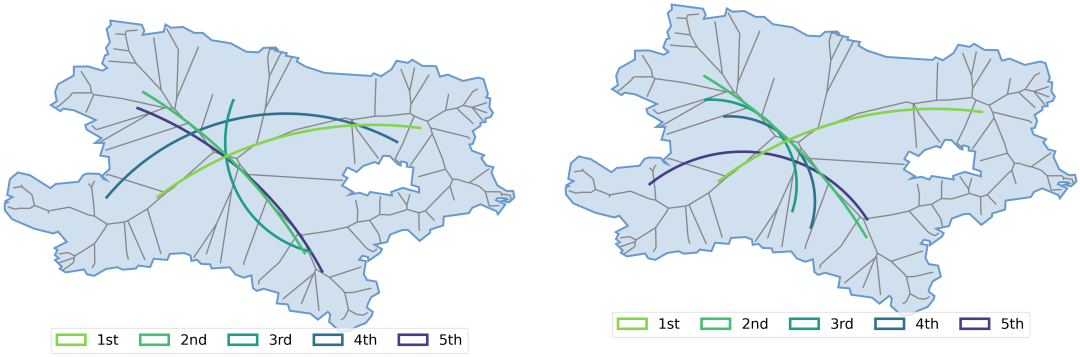
A notable drawback of this enhancement is the computational cost, as it requires recalculating the perceived coverage for each arc in the neighborhood in every iteration.



(a) Salzburg without clustering.

(b) Salzburg, with agglomerative clustering.

Figure 6.8: Lower Austria clustering comparison with selection based on Q with $w_{cov} = 1.0, w_{cur} = 0.25, w_{ori} = 0.1$.



(a) Lower Austria, without clustering.

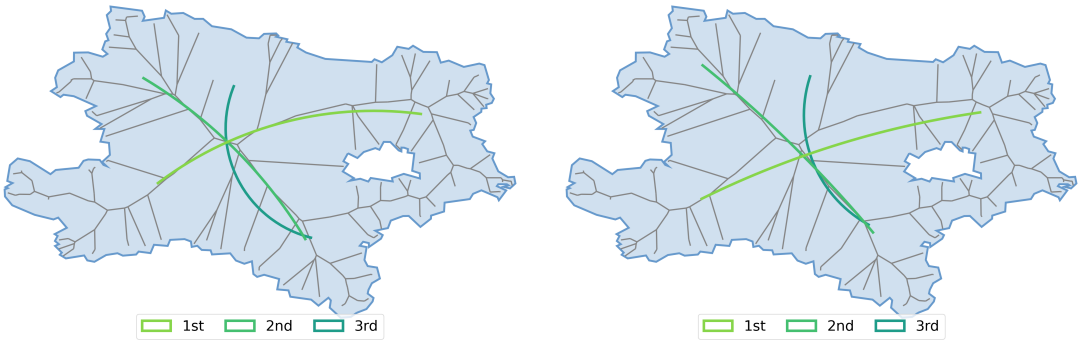
(b) Lower Austria, with agglomerative clustering.

Figure 6.9: Lower Austria clustering comparison with selection based on Q with $w_{cov} = 1.0, w_{cur} = 0.25, w_{ori} = 0.1$.

As a result, the runtime of the algorithm increases substantially (see the next section). Therefore, this step is recommended for refining only a small number of arcs.

6.3 Runtime

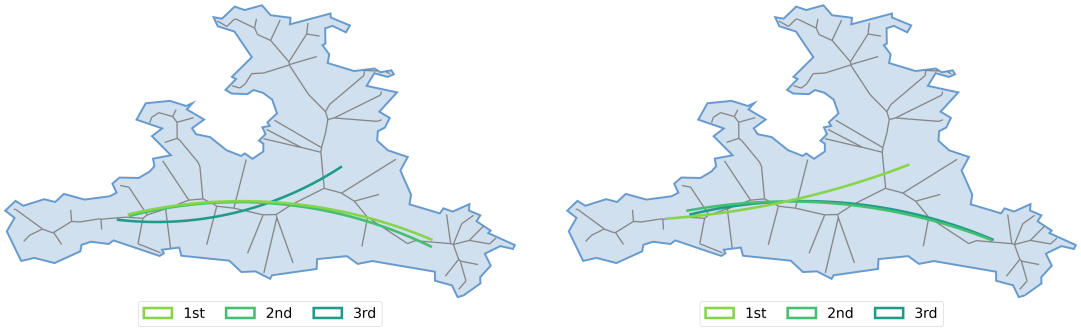
Although runtime was not a primary focus of this thesis, we present some results here along with potential improvements that could enhance the performance. In Figure 6.12, we can observe the runtime results based states of on the Austrian dataset. For polygons with fewer than 500 vertices, label placements could be achieved in 1–2 seconds, while polygons with higher vertex counts required more time. Thus, the algorithm is well suited for guiding cartographers in the placement of labels, but may not be completely sufficient for an interactive setting on areas with very high vertex counts. However, for maps with multiple areas, the labeling could be easily parallelized, since the label placement can



(a) Top 3 positions before the local search has been applied.

(b) Top 3 positions after the local search has been applied.

Figure 6.10: Post-processing using local search with a maximum of 10 iterations, Lower Austria, top 3 positions, label height = 0



(a) Top 3 positions before the local search has been applied.

(b) Top 3 positions after the local search has been applied.

Figure 6.11: Post-processing using local search with a maximum of 10 iterations, Salzburg, top 3 positions, label height = 0

be carried out independently for each area. By performing a local search on the top 5 candidates, the runtime additionally increased by approximately 50% to 200%, to 4–16 seconds.

The primary bottleneck of the implementation lies in the intersection calculations, which consume a significant portion of the runtime; more than 80% depending on the vertex count. The intersection calculations are done in two areas. First, during the position generation step, where we calculate the intersections of the generated circle with the polygon. And secondly, during the perceived coverage calculation for the $pc(s)$ values, recall Section 5.1.

A possible issue lies in the use of a simple array of segments to compute the intersections, which requires calculating intersections for each polygon segment individually. A solution for this would be to use a tree-like data structure that is better suited for intersections,

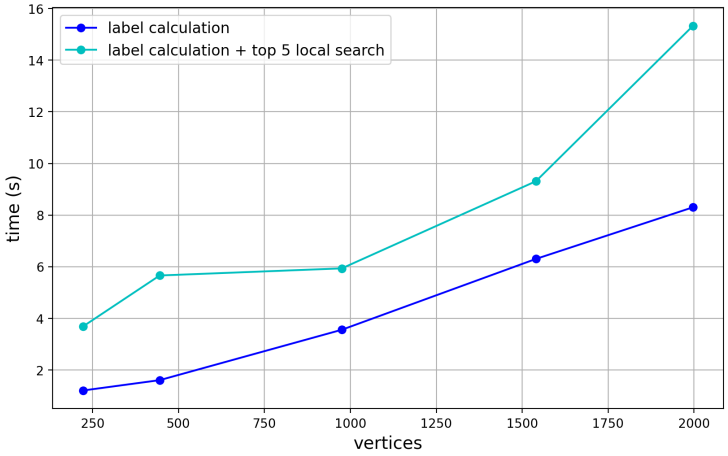


Figure 6.12: HeightConstrainedLabeling algorithm runtime on 30 generated paths, averaged over 30 runs.

since it reduces the amount of intersection-checks significantly. Additionally, we could apply morphological erosion in order to simplify the polygon while keeping the characteristics that are necessary for labeling. This would also substantially reduce processing times.

CHAPTER 7

Conclusion

In this thesis, we explored the area labeling problem, focusing specifically on methods for placing labels along circular arcs while ensuring conformity. To this end, we presented and compared the approaches of Barrault [Bar01] and Krumpe [Kru20a]. Both of which follow a pipeline structure consisting of position generation and position selection steps. After discussing the methods and identifying their weaknesses, we additionally proposed the HeightConstrainedLabeling problem variant as well as a labeling pipeline implemented in C++.

In labeling, it is essential to consider potential overlaps with other labels and map features. To minimize conflicts, a diverse set of label candidates is necessary. Through experiments, we could confirm that Krumpe's position generation is superior to Barrault's, since it can generate a more diverse candidate set. Additionally, Barrault's generation method does not consider label height. Therefore, to ensure that labels of a specific height remain within the boundary, HeightConstrainedLabeling introduces a height parameter.

In the position selection step, we presented Barrault's approach, which is based on his perceived coverage metric. However, as noted by Barrault himself, this metric does not account for readability criteria like label curvature and orientation. To address this, we introduced a quality function that integrates these additional metrics, which we believe significantly improves the overall placement quality. To increase diversity among the top paths, we incorporated clustering in the position selection step. As a post-processing step, we added local search to further improve the top paths.

We also implemented the proposed enhancements in C++ and visualized the results using Matplotlib. Our implementation addresses some of the weaknesses in Barrault's and Krumpe's approaches, but it also has its own limitations. Firstly, similarly to Barrault's method, our implementation may result in generated labels overlapping holes within the polygon if those holes are sufficiently small for the labels to completely enclose them. Additionally, when using clustering, the cluster amount should be approached with some

consideration, since it has a significant impact on the clustering quality. While runtime was not a primary focus of this thesis, it could be further improved in the future to make the implementation more suitable for interactive use on larger maps.

As the amount of information presented on maps continues to grow, high-quality labeling will become even more important to warrant further research. It would be interesting to integrate methods that ensure aesthetic continuity between multiple labels on a map. This could improve the visual coherence and make maps more visually appealing. Additionally, by implementing a more advanced clustering method, we could achieve an even more diverse suited label candidates.

Overview of Generative AI Tools Used

I used Grammarly and ChatGPT to check grammar, rephrase individual sentences and find synonyms.

List of Figures

List of Algorithms

Bibliography

[Bar01] Mathieu Barrault. A methodology for placement and evaluation of area map labels. *Computers, Environment and Urban Systems*, 25(1):33–52, 2001. Publisher: Elsevier.

[CGA24] The Computational Geometry Algorithms Library CGAL. `https://doc.cgal.org/5.6.1/Manual/index.html`, February 2024. Last accessed: 07.10.2024.

[Cue72] René. Cuenin. *Notions générales et principes d'élaboration*. Collection scientifique de l'Institut géographique national. Eyrolles, 1972.

[DBCVKO08] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DF92] Jeffrey S. Doerschler and Herbert Freeman. A rule-based system for dense-map name placement. *Commun. ACM*, 35(1):68–79, January 1992.

[DKSW02] Steven Van Dijk, Marc Van Kreveld, Tycho Strijk, and Alexander Wolff. Towards an evaluation of quality for names placement methods. *International Journal of Geographical Information Science*, 16(7):641–661, November 2002. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/13658810210138742.

[DPP03] Dirk Dörschlag, Ingo Petzold, and Lutz Plümer. Placing objects automatically in areas of maps. January 2003.

[ECMS96] Shawn Edmondson, Jon Christensen, Joe Marks, and Stuart Shieber. A General Cartographic Labelling Algorithm. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 33(4):13–24, December 1996.

[FA87] Herbert Freeman and John Ahn. On the problem of placing names in a geographic map. *International Journal of Pattern Recognition and Artificial Intelligence*, 01(01):121–140, April 1987. Publisher: World Scientific Publishing Co.

[H67] Blum H. A transformation for extracting new descriptions of shape. *Models for the perception of speech and visual form*, pages 362–380, 1967. Publisher: MIT Press.

[Imh75] Eduard Imhof. Positioning Names on Maps. *The American Cartographer*, 2(2):128–144, January 1975. Publisher: Taylor & Francis _eprint: https://doi.org/10.1559/152304075784313304.

[Kru20a] Filip Krumpe. *Labeling interactive maps*. doctoralThesis, 2020. Accepted: 2021-05-12T08:12:33Z ISBN: 9781757716642.

[Kru20b] Krumpe, Filip. Area Labeling. `https://gitlab.vgiscience.de/map_labeling/area_labeling/area_labeling`, February 2020. Last accessed: 07.10.2024.

[LDLD19] Fuyu Lu, Jiqiu Deng, Shiyu Li, and Hao Deng. A hybrid of differential evolution and genetic algorithm for the multiple geographical feature label placement problem. *ISPRS Int. J. Geo Inf.*, 8:237, 2019.

[Lee82] D. T. Lee. Medial Axis Transformation of a Planar Shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):363–369, July 1982. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

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

[OH24] Rachid Oucheikh and Lars Harrie. A feasibility study of applying generative deep learning models for map labeling. *Cartography and Geographic Information Science*, 51(1):168–191, January 2024. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/15230406.2023.2291051.

[Ope24] Statistik Austria Open.data. `https://data.statistik.gv.at/web/meta.jsp?dataset=OGDEXT_GEM_1`, July 2024. Last accessed: 07.10.2024.

[PF96] I. Pinto and H. Freeman. The feedback approach to cartographic areal text placement. In Petra Perner, Patrick Wang, and Azriel Rosenfeld, editors, *Advances in Structural and Syntactical Pattern Recognition*, pages 341–350, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[TC89] Samuel M Thomas and Y. T Chan. A simple approach for the estimation of circular arc center and its radius. *Computer Vision, Graphics, and Image Processing*, 45(3):362–370, March 1989.

[WDZL16] Changbin Wu, Yuan Ding, Xinxin Zhou, and Guonian Lu. A grid algorithm suitable for line and area feature label placement. *Environmental Earth Sciences*, 75(20):1368, October 2016.

[Yoe72] Pinhas Yoeli. The Logic of Automated Map Lettering. *The Cartographic Journal*, 9(2):99–108, December 1972.