Generating Optimal Curves for Necklace Maps

Brocken, Thomas

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Applied Geometric Algorithms group

Generating Optimal Curves for Necklace Maps

Master's Thesis

Thomas Brocken

Supervisors:
prof.dr. B. Speckmann
dr. K.A.B. Verbeek

Committee:
prof.dr. B. Speckmann
dr. K.A.B. Verbeek
dr.ir. J.J.A. Keiren

Eindhoven, July 2020

Abstract

As more and more events play out on a global scale, the need for good visualization of geographical data rises. Speckmann and Verbeek [31] introduced a novel type of thematic map called necklace maps. They also present an algorithm that computes suitable symbols given a necklace. Thus far, no algorithms exist that generate the necklaces. In this thesis we develop algorithms that generate suitable necklaces for necklace maps. Our algorithms produce two types of necklaces: circular necklaces and star-shaped necklaces. A suitable circular necklace satisfies a few quality criteria: (1) The underlying map should be recognizable. (2) It should be easy to relate the data to the corresponding regions. (3) The necklace should be as small as possible while surrounding the regions. For star-shaped necklaces, we additionally identify the following quality criterion: (4) The necklace should follow the shape of the set of regions it surrounds closely.

Our algorithms take as input a set of polygons that represent the regions of the map. For circular necklaces we first approximate the regions as a set of colored points, and then compute the smallest disk that contains at least some number of points of a certain color, for every color. Our algorithm finds this disk in expected $O(n^2 \log n)$ time, where n is the total number of points. For star-shaped necklaces we first compute the union of all polygons. Then, we compute a small set of feature points, using the medial axis, that describe the shape of the continent. Finally, we draw a cubic Bézier spline through the feature points. We then interpolate between circular necklaces and star-shaped necklaces. The algorithms are experimentally evaluated on different maps. In general the results are visually pleasing.

Contents

Chapter 1

Introduction

In this day and age, society is driven by data. As more and more events play out on a global scale, the need for good visualization of geographical data rises. A good example is the COVID-19 pandemic during which all sorts of maps were used to visualize the spread of the virus. These maps visualize data that have a non-negative numeric value for every region on the map. One such type of map is the *proportional symbol map*, see Figure 1.1(a). A proportional symbol map is a type of *thematic map*, that is, a map that shows a particular theme using geography as a reference. It uses symbols of varying size to represent data values. These symbols are scaled such that the size of the symbol is proportional to the data value of the region it belongs to. The symbols are typically disks or squares, but any shape is possible. Proportional symbol maps look good when there are not too many regions or when small regions have a small data value associated with them. However, as the number of regions increases, the map becomes cluttered and difficult to parse. Symbols can also occlude regions entirely—in Figure 1.1(a) the Benelux is not visible anymore. To solve these issues, Speckmann and Verbeek [31] presented a new type of thematic map called necklace maps. The general idea of necklace maps, see Figure 1.1, is to place symbols of varying size—also called *beads*—along a curve called the *necklace*. The beads of the necklace display data values associated with the regions, similar to proportional symbol maps. This necklace goes along the outside of the map. By placing symbols on the necklace instead of directly on the regions, less of the map will be occluded. However, this makes it harder to associate the symbols to their corresponding regions.

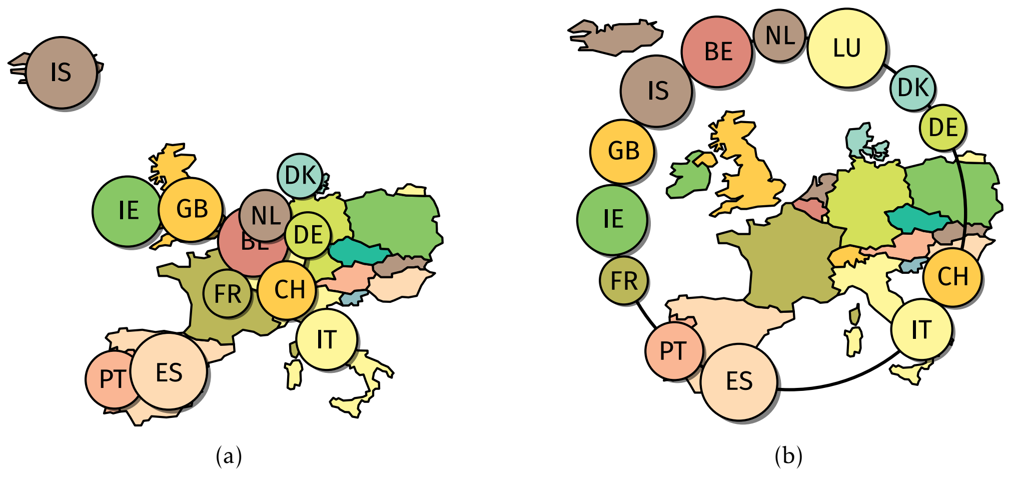

Figure 1.1: Total number of COVID-19 cases per 1 million population as of June 11th, 2020. Left a proportional symbol map, right a necklace map with a circular necklace denoted by the black line.

An advantage of necklace maps compared to proportional symbol maps is that the symbols in necklace maps are still discernible even if small regions have a large data value. For example, the proportional symbol map in Figure 1.1(a) contains overlapping symbols which make it hard to read; the necklace map in Figure 1.1(b) does not have this problem. Another advantage is that the linear ordering of the beads along the necklace makes it easier for the reader to compare data values. A challenge for necklace maps is that it should be easy to associate symbols to their corresponding regions. A specific styling can help mitigate this issue: We give the beads the same color as the corresponding region, and we place the name of the region on the beads. A good necklace is also important. It is easier to associate a symbol to its corresponding region if the symbol lies close to the region. In order to place all symbols close to their corresponding regions, we need a necklace that closely surrounds the regions. However, this necklace should also not lie too close to the regions, because then the symbols might occlude the regions.

Speckmann and Verbeek present an algorithm [32] that, given a necklace, computes optimal symbols with relation to a a few quality criteria: (1) It should be easy to associate the symbols to their corresponding regions. (2) The symbols should be disjoint. (3) The symbols should be maximal in size. (4) The symbols should be suitably ordered along the necklace. This algorithm also works with multiple, disjoint or nested, necklaces. Their algorithm does not generate necklaces, nor does any algorithm currently exist that generates necklaces. However, we do want to be able to generate necklaces automatically. This problem is challenging because we need something precise to optimize for, while human intuition on what makes a necklace good, is vague and unclear. Besides the size and the location of the necklace, the shape also plays a significant role. Circles are perhaps the most visually pleasing shape for necklaces. However, where a circular necklace looks good on one map, it might not work so well for another map, see for example Figure 1.2. In this figure the circular necklace leaves a significant amount of empty space in the bottom left quadrant and can make a symbol corresponding to Nigeria lie far away from Nigeria itself. Here we should use a necklace that follows the shape of the continent more closely.

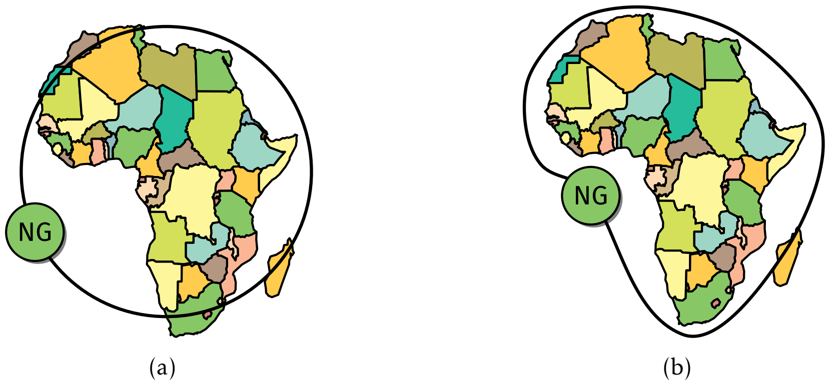

Figure 1.2: (a) A circular necklace is shown which leaves a significant amount of empty space in the bottom left quadrant of the necklace. (b) A freeform necklace, which is better suited for this map.

The goal of this thesis is to develop an algorithm that generates suitable necklaces with relation to a few quality criteria. We consider two types of necklaces: one where the necklace must be a circle, and one where the necklace can take a freeform shape. For this we split the algorithm into two separate parts. Algorithm CircularNecklace produces a suitable circular necklace, and algorithm FreeformNecklace produces a suitable freeform necklace.

What makes a necklace suitable is captured in the following quality criteria.

- Recognizability – Because necklace maps aim to solve, among others, the problem of occlusion that proportional symbol maps have, the underlying map should not be occluded too much. One should be able to easily relate the data to the corresponding regions, but for that the regions need to be recognizable. Therefore, occlusion is allowed but important borders should be visible. Hence, generally we want the necklace to surround all regions.
- Position – It should be easy to associate the symbols to their corresponding regions. The algorithm by Speckmann and Verbeek [32] places the symbols as well as possible for any given necklace. However, a good necklace helps as well, as shown in Figure 1.2.
- Size – The necklace should be as small as possible. Assuming the necklace surrounds the regions, this allows balance between the size of the symbols and the size of the regions. It also makes the symbols lie closer to the regions, which makes the map easier to read.

For freeform necklaces, we additionally identify the following quality criterion.

- Shape – The necklace should follow the shape of the set of regions it surrounds closely. Again, the goal is to allow the reader to easily associate the symbols to their corresponding regions.

1.1 Problem Statement

We want to compute a suitable necklace for a given input map. We formalize the problem as follows. Let $\{P_1,...,P_m\}$ be a set of m polygons which represent the regions of the input map. A polygon is a shape described by a finite number of straight line segments that form a closed circuit. The line segments are called edges and the points connecting the edges are vertices. A *simple polygon* is a polygon that does not have holes and does not intersect itself. A *convex polygon* is a simple polygon in which every line segment between two points on the boundary lies completely within the polygon. We split the problem into two parts: compute a suitable circular necklace, and, compute a good freeform necklace. We will now describe the output, requirements, and modeling for CircularNecklace and FreeformNecklace.

Circular necklaces. The output of CircularNecklace is a circle, which we model as a tuple (p,r) where p is the center of the circle and r is the radius of the circle. One of the quality criteria for circular necklaces is that occlusion is allowed, but the underlying map should not be occluded too much. Occlusion is caused by the symbols. In Figure 1.3 part of a necklace is shown together with its *annulus*. An annulus is a region bounded by two concentric circles. The annulus of a necklace is the region in which the symbols lie. Since the annulus of the necklace is determined when constructing the symbols—which happens in the algorithm by Speckmann and Verbeek—we do not have access to it at this stage. Hence, in order to capture occlusion, we should place the necklace in such a way that the annulus of the necklace will not occlude the regions too much. We can solve this by placing the necklace around all regions. However, placing the necklace around all regions might result in a necklace that is too large. Hence we allow some overlap between the interior of the necklace and the regions. Since a smaller region is more easily occluded by the annulus than a larger region, we should be able to specify how much of a region the necklace is allowed to overlap. To this end we model the amount of overlap between the necklace and the regions as proper fractions, that is, a fraction that lies between 0 and 1. Every region in the map is assigned a fraction that dictates what fraction of the area should lie within the circle. For a polygon P_i and its assigned fraction $0 \leq x_i \leq 1$, at least x_i times the area of P_i should lie within the circle. This model allows us to give large regions a small fraction and small regions a large fraction. We can also give a region the fraction 0, then the region does not have to lie within the circle. We use this flexibility to improve readability, by, for example, placing Iceland outside

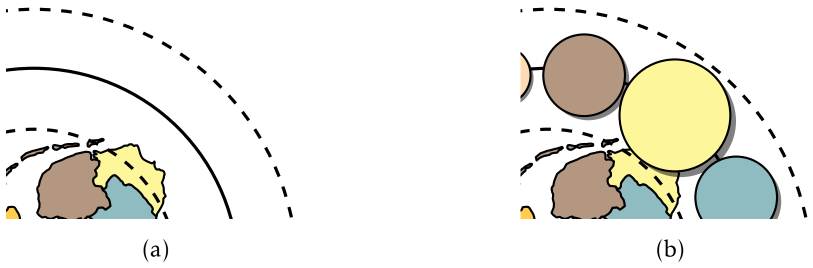

Figure 1.3: (a) An annulus (dashed lines) together with a necklace (solid line). (b) The width of the annulus represents the maximum size a symbol can have.

of the necklace and giving Russia a smaller fraction, see Figure 1.4. We optimize the radius of the circle by making it as small as possible while satisfying the fractions of each polygon. This satisfies our third quality criterion. Finding the smallest disk that satisfies the fractions for all polygons, is a challenging problem. The objective is to find that smallest disk that contains at least some fraction of the area of a certain polygon, for every polygon. To reduce complexity, we transform the problem into an optimization problem on a set of colored points. We transform each polygon into a set of points that represents the area of the polygon well. How exactly we pick this set of points can be read in Chapter 2. Every set of points representing each polygon is given its own color. For a set of n colored points of m different colors, the optimization problem is then to find the smallest disk that contains at least some number of points—according to the fractions—of a certain color, for every color. We solve this problem in Chapter 3.

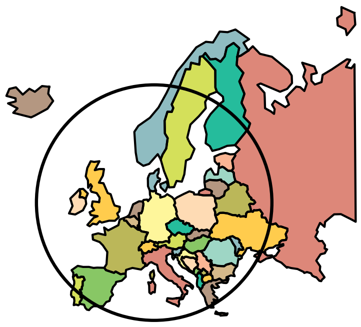

Figure 1.4: A circular necklace. Iceland has been excluded and just a small fraction of the area of Russia lies within the necklace.

Freeform necklaces. The output of FreeformNecklace is modeled as a set of connected *Bézier curves* called a *Bézier spline*. Bézier curves are parametric equations that were originally introduced by Paul de Casteljau [9].

A requirement from the algorithm that places symbols on the necklace by Speckmann and Verbeek [32], is that the necklace should be a *star-shaped* curve, that is, a curve within which there exists a point that can see the whole curve. This curve should be smooth. In particular, it should be a C^1-continuous curve, that is, a curve of which the first derivative is continuous, thus no sharp bends may occur in the curve, see Figure 1.5. From this point onward, we will refer to freeform necklaces as *star-shaped necklaces*.

To satisfy the additional quality criterion for star-shaped necklaces, we should be able to capture the global shape of the set of regions. We do this by modeling the global shape of the polygon as a set of points called *feature points*. These feature points are placed in strategic positions on the border of the set of regions. We can apply offsets to the feature points in order to change the amount of overlap the necklace has with the regions. How we construct the set of feature points and use them to compute a suitable Bézier spline can be read in Chapter 4.

Figure 1.5: (a) A continuous curve containing a sharp bend, making it C^0-continuous. (b) A (smooth) continuous curve with a continuous derivative, making it C^1-continuous.

1.2 Related Work

Thematic maps. There exist many types of thematic maps for visualizing per-region data. The choropleth map [40] is perhaps the most frequently used type of thematic map used to display geospatial data. Choropleth maps make it very easy to relate data to the corresponding regions—the regions are simply colored or shaded. However, this benefit comes with a few downsides. Unless a legend is used, it is hard to compare the data value of two differently shaded regions. Large regions also tend to be overemphasized and their shading might suggest that the data is uniformly distributed throughout the region. A choropleth map works better when the regions are similar in size.

Another popular type of thematic maps is the cartogram. This type of map distorts the regions such that the area of the region is proportional to the data value corresponding to that region. Cartograms are good at showing large discrepancies between regions, however, it may become difficult to compare small regions, depending on the variance between the data values. Algorithms that compute cartograms have existed for some time [10, 34, 37].

A third popular type of thematic maps is the proportional symbol map—also called *graduated symbol map*. Proportional symbol maps make use of symbols placed on regions of which the size is proportional to the data value of that region. This makes it easy to associate symbols with regions. It also conveys quantities better than choropleth maps. A downside is that these symbols may overlap when there are many small regions, making the map hard to read. Large symbols can also occlude small regions. Cabello et al. explore how overlapping symbols should be stacked to achieve optimal readability [7].

Necklace maps. Necklace maps are a novel type of thematic maps introduced by Speckmann and Verbeek [31]. As we touched on before, necklace maps place symbols on a curve around the regions called the necklace. Necklace maps solve the problem of occlusion that proportional symbol maps have. However, by placing the symbol on a curve, the ability to associate symbols to their corresponding regions is hindered. The algorithms [32] by Speckmann and Verbeek compute, given a necklace, a necklace map with optimally placed symbols, where optimal refers to a few quality criteria the authors define: (1) It should be easy to associate the symbols to their corresponding regions. (2) The symbols should be disjoint. (3) The symbols should be maximal in size. (4) The symbols should be suitably ordered along the necklace. Every symbol has a corresponding feasible interval on the necklace which specifies acceptable positions for that symbol. The authors first compute feasible intervals on the necklace, they then optimize the symbol sizes, and finally they optimize the symbol placements. They also extend their algorithm to work with nested necklaces and with multiple disjoint necklaces.

Smallest enclosing disk. Since we would like to solve a problem on colored points, we first explore the problem space of enclosing points using a circle. Problems related to finding the smallest disk containing some number of points have been studied for quite some time. In 1983 Megiddo presented an algorithm that finds the smallest circle containing a set of n points in $\mathbb{R}^2$ in $O(n)$ time [24]. In 1993 Efrat et al. [12] presented an algorithm that solves the SMALLEST k-ENCLOSING DISK problem. In this problem the objective is to find the smallest disk that contains at least k points from a set of n points. They solve this problem in $O(nk \log^2 n)$ time using $O(nk)$

storage. In 1995 Matoušek improved upon this bound and presented a randomized algorithm that finds the minimal disk that encloses at least k points in expected $O(n \log n + nk)$ time [21]. More recently, Har-Peled et al. improved upon the algorithm of Matoušek and presented a randomized algorithm that runs in $O(nk)$ expected time [18].

A different kind of problem, in which we introduce colors, is the problem of finding the Smallest Color-Spanning disk. This problem is about finding the disk with smallest radius that contains at least one point of each of the m colors in total. It can be found using the upper envelope of m Voronoi surfaces. A *Voronoi diagram* of a set of points in $\mathbb{R}^2$—called sites—is a partition of the plane into Voronoi cells such that all points in a cell have the same nearest site. The Voronoi surface of a Voronoi diagram is the set $\{(p, d(p)) \mid p \in \mathbb{R}^2\} \subseteq \mathbb{R}^3$, where $d(p)$ is the minimum distance from p to any site of the Voronoi diagram. The upper envelope of m Voronoi surfaces can be computed in $O(nm \log n)$ time and has complexity $\Theta(nm)$ [20]. Abellanas et al. present an algorithm that computes the farthest color Voronoi diagram, from which the smallest color-spanning disk can be computed, in $O(n^2 \alpha(m) \log m)$ time [1], where α is the inverse Ackermann function.

The Exact Colored k-enclosing disk problem is a combination of the problems described above. In this problem the objective is to find a disk that encloses exactly some amount of points of a certain color, for every color. This problem has recently been studied by Barba et al. [4]. The algorithm presented by Barba et al. makes use of higher order Voronoi diagrams, specifically a *kth order Voronoi diagram*, to find such a disk. A kth order Voronoi diagram is a special type of Voronoi diagram where every cell consists of points that have the same k nearest sites. Their algorithm runs in $O(KVD(n,k))$ time where $KVD(n,k)$ denotes the time required to construct the kth order Voronoi diagram. An efficient algorithm for computing a kth order Voronoi diagram is presented by Agarwal et al. Their algorithm runs in $O(k(n-k)\log n + n \log^3 n)$ expected time [2]. Barba et al. also extend this problem to the Smallest Exact Colored k-enclosing disk problem and mention that the smallest disk can be found with at most an $O(k)$ increase in running time by computing the smallest enclosing disk of the k nearest neighbors for every candidate Voronoi cell. The problem we would like to solve, the Smallest Colored k Enclosing Disk problem, is closely related. We want to find the smallest disk that contains *at least* some number of points of a certain color, for every color. This problem is similar to the problem that Barba et al. solved, however, we do not require an exact number of points.

Curved schematization. We want to draw curves around continents. Hence, we first explore the problem space of drawing curves with relation to geography. The process of *schematization* is to greatly reduce detail of maps beyond what is necessary to achieve its goal. A well-known example are the metro maps. *Curved schematization* aims to schematize maps using curves. Fink et al. [13] use *cubic* Bézier curves to draw metro maps automatically. A cubic Bézier curve is a curve given by a cubic polynomial and is drawn using four control points. Mi et al. [25] provide a computational method for the abstract depiction of 2D shapes in terms of parts. Curved schematiztaion has been explored by Van Goethem et al. [36]. Their algorithm transforms polygons into a schematic representation using circular arcs. In another paper, Van Goethem et al. [35] present a framework for topology-preserving curved schematization.

1.3 Contribution and Organization

The remainder of the thesis is structured in the following way. In Chapter 2 we transform polygons into representative sets of colored points in order to transform a problem on polygons into a problem on sets of colored points. In Chapter 3 we explore various algorithmic ideas and we present an algorithm that finds the smallest disk containing at least some number of points of a certain color, for every color. In Chapter 4 we present a method to compute suitable star-shaped necklaces using Bézier curves. Finally, we experimentally evaluate our algorithms and models in Chapter 5. The experiments show that our results are visually pleasing.

Chapter 2

Discretizing Polygons

In this chapter we discuss how we can transform a polygon into a strategically chosen set of points that captures the area of the polygon well. This set of points should be chosen in such a way that we also capture small features such as protrusions and jagged edges. One way to do this, see Figure 2.1, would be to overlay the polygon with a very fine grid of high enough resolution. However, one can imagine that we require a very fine grid in order to capture these small features. This may require many points. In Figure 2.1(a) mainland Greece is shown. In Figure 2.1(b) we overlay this region with a dense grid of points. However, as we show in Figure 2.1(c), even a dense grid might not represent the area of the polygon well. Thus an even finer grid is required. Moreover, the inner region of Figure 2.1(b) contains many points that are superfluous. As we have seen in Chapter 1, algorithms for colored points are inefficient for large numbers of points, and hence using a small number points is desirable. This brings us to the problem statement for this chapter: How can we sample a small set of points from a polygon and prove that this sample represents the area polygon well enough? More precisely, we require that a disk that covers 1/2 of the points sampled from a polygon, covers between $1/2-\varepsilon$ and $1/2+\varepsilon$ of the area of the polygon, for an error bound ε. We are interested in how many points we should sample to satisfy these error bounds.

We would like to approximate the area of a polygon $\mathcal{P}$ using a small set of points. Suppose P is a set of points that perfectly captures the area of $\mathcal{P}$—imagine a grid of points in $\mathcal{P}$ whose density approaches infinity. We can then use P as a proxy for the area of $\mathcal{P}$. This allows us to approximate the area of $\mathcal{P}$ by approximating P using a subset A of P. Then, if A captures P well, A will also capture the area of $\mathcal{P}$ well. A suitable set A can be constructed using the notion of an ε-approximation.

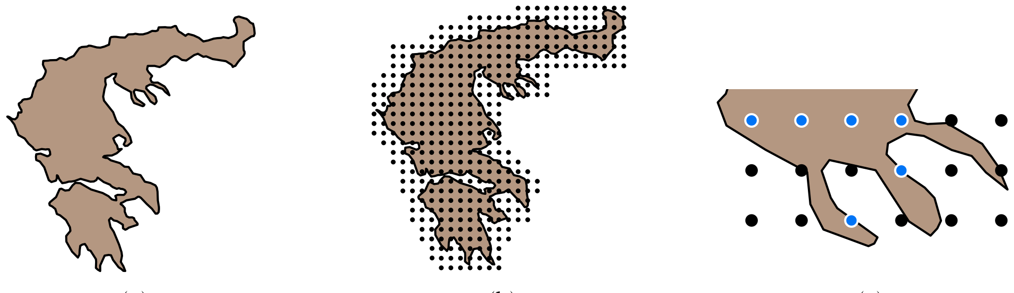

Figure 2.1: (a) Mainland Greece. This polygon has many jagged edges and thin protrusions. We need a very fine grid in order to capture all the features. (b) A fine grid overlaid on mainland Greece. (c) This area of Greece—*Chalkidiki*—is not captured well, even for our fine grid. The blue points are the only points that lie within the polygon.

2.1 ε-Approximations

An ε-approximation, introduced by Vapnik and Chervonenkis [39], of a set of points P is a subset A of P that captures P with respect to a set of geometric objects, such as disks. The value of ε is the error we are willing to accept: If some disk covers $1/2$ of the points of A, then that disk covers between $1/2 - \varepsilon$ and $1/2 + \varepsilon$ of the points of P and vice versa. Haussler and Welzl introduced a useful set of notations [19] to describe the notion of ε-nets, a notion related to ε-approximations. We use their notations when defining ε-approximations. Let X be a set of points and let $\mathcal{R}$ be a family of subsets of X called *ranges*. A *range space* S is a pair $(X, \mathcal{R})$. For some finite subset P of X, $\mathcal{R}|_P$ is the *projection* of $\mathcal{R}$ onto P, which is defined as

$$\mathcal{R}|_P := \{P \cap R \mid R \in \mathcal{R}\}$$

P is *shattered* by $\mathcal{R}$ if $\mathcal{R}|_P$ is equal to the powerset of P, that is, the set of all subsets of P. The *Vapnik–Chervonenkis (VC) dimension* of $(X, \mathcal{R})$ is the cardinality of the largest subset of X that can be shattered by $\mathcal{R}$. The notion of VC dimension was first introduced by Vapnik and Chervonenkis [39] and is used in statistical learning theory (see for example [38]) and computational geometry.

The VC dimension of disks. Consider the range space $S = (\mathbb{R}^2, \mathcal{D})$ with $\mathcal{D}$ being the family of all disks in $\mathbb{R}^2$. Let $P = \{p_1, p_2, p_3\} \subseteq \mathbb{R}^2$. The powerset of P is

$$\{\emptyset,\ \{p_1\},\ \{p_2\},\ \{p_3\},\ \{p_1, p_2\},\ \{p_1, p_3\},\ \{p_2, p_3\},\ \{p_1, p_2, p_3\}\}$$

Clearly we can cover $\{p_1, p_2, p_3\}$ with one disk. We can cover any combination of two points of P by the three disks shown in Figure 2.2(a). Lastly, we can cover every point of P separately by placing a disk around every point. We can also cover the empty set. We conclude that any set P with at most three points can be shattered by $\mathcal{D}$.

Let $Q = \{q_1, q_2, q_3, q_4\} \subseteq \mathbb{R}^2$. We can distinguish between two cases: either three points lie on the convex hull of Q, or all four points lie on the convex hull of Q. In the first case any disk that covers the three points on the convex hull, also covers the fourth point. In the second case, four points lie on the convex hull. Let q_1, q_2, q_3, q_4 be four points ordered along the boundary of the convex hull. We can either cover $\{q_1, q_3\}$, or $\{q_2, q_4\}$ with a disk, but not both at the same time.

Suppose for contradiction that there exist two disks D_1, D_2 that cover $\{q_1, q_3\}$ and $\{q_2, q_4\}$ separately. Then edges $\overline{q_1 q_3}$ and $\overline{q_2 q_4}$ are in the Delaunay graph of Q [5, Theorem 7.4(ii)]. The Delaunay graph of a planar point set is a *plane graph* [5, Theorem 9.5], that is, a graph in which no edges cross. This contradicts with the fact that two crossing edges, $\overline{q_1 q_3}$ and $\overline{q_2 q_4}$, are in the Delaunay graph of Q. Thus there do not exist two disks D_1, D_2 that cover $\{q_1, q_3\}$ and $\{q_2, q_4\}$

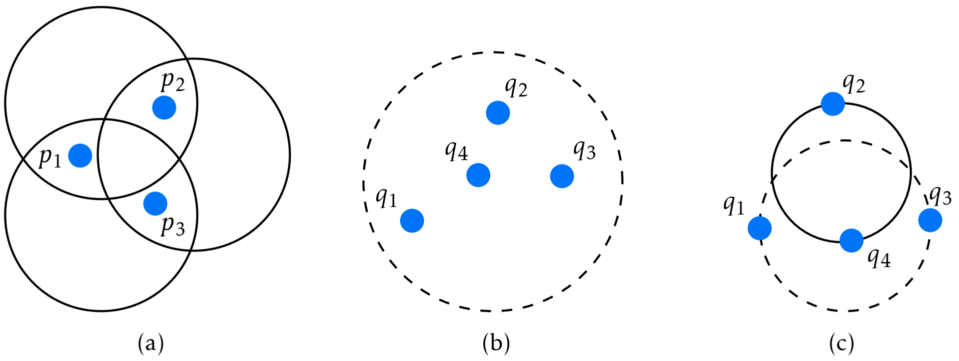

Figure 2.2: (a) A set of 3 points of which every pair can be covered separately. (b) A set of 4 points. We can not find a disk that covers just $\{q_1, q_2, q_3\}$. (c) A set of 4 points. We can not find a disk that covers just $\{q_1, q_3\}$

separately—see Figure 2.2(c) for an example of this case. Since sets containing four points can not be shattered, we conclude that the VC dimension of $(\mathbb{R}^2, \mathcal{D})$ is three.

ε-approximations. We use definitions from the book by Mustafa and Varadarajan [26]. Let $S = (X, \mathcal{R})$ be a range space, let P be a finite subset of X, and let $A \subseteq P$ be a uniformly random sample of P. A range $R \in \mathcal{R}$ is *well-represented* in A if $|\frac{|R \cap P|}{|P|} - \frac{|R \cap A|}{|A|}| \leq \varepsilon$. This implies that if R intersects $1/2$ of the points of A, it also intersects between $1/2 - \varepsilon$ and $1/2 + \varepsilon$ of the points of P. A is an ε-approximation for $\mathcal{R}$ if and only if every $R \in \mathcal{R}$ is well-represented in A. Parameter $0 < \varepsilon \leq 1$ denotes the maximum error between the actual fraction and the approximated fraction of the number of points intersected by a range. For $\varepsilon = 1$, any nonempty set $A \subseteq P$ is an ε-approximation for $\mathcal{R}$. Vapnik and Chervonenkis proved the following theorem.

Theorem 2.1 ([39]). *Let $(X, \mathcal{R})$ be a range space of VC dimension d. For a finite set of points $P \subset X$, $\varepsilon > 0$, $\delta \leq 1$ there exists a sufficiently large constant c such that for a random sample A of P of cardinality at least*

$$\frac{c}{\varepsilon^2}(d \log \frac{d}{\varepsilon} + \log \frac{1}{\delta})$$

is an ε-approximation of P for $\mathcal{R}$ with probability at least $1 - \delta$.

Thus for $\delta < 1$ there exists an ε-approximation for $\mathcal{R}$ of size

$$O(\frac{d}{\varepsilon^2} \log \frac{d}{\varepsilon} + \frac{1}{\varepsilon^2} \log \frac{1}{\delta})$$

Remarkably, the bound on the size of ε-approximations does not depend on the size of P. This allows us to sample a constant number of points from a polygon, regardless of the size of that polygon. In theory, this theorem dictates that we should pick a large number of points, even for small values of c—in the hundreds for $\varepsilon = 0.1$, $c = 1$, $\delta = 0.5$. In practice however, we will pick a much smaller number of points.

Note that Theorem 2.1 requires a finite set of points P. However, in theory one could sample an infinite number of points from a polygon. We sample directly from the polygon by sampling as if we work in the limit of the size of the grid. However, a computer only has limited precision, thus a random number generator provides discrete finite numbers. Hence, in practice we satisfy Theorem 2.1 since we can only sample a finite number of points from the polygon.

2.2 Sampling Points

In this section we will show how we can sample points from a polygon. In order to do that we first show how one can sample a point from a triangle. We then extend this to an entire polygon. We sample a uniformly random point from a triangle $\triangle ABC$—see Figure 2.3—as follows. We generate two random numbers x, y from $U[0,1]$ where $U[0,1]$ is a uniform distribution of real numbers between 0 and 1. We then define our random point p as $p = x \cdot \overrightarrow{AB} + y \cdot \overrightarrow{AC}$. If $A + p$ lies inside of $\triangle ABC$, then $A + p$ is our randomly sampled point. Otherwise, $A + p$ falls inside triangle $\triangle BCD$, the triangle which forms a parallelogram together with $\triangle ABC$—Figure 2.3(b)—and can be mapped back onto triangle $\triangle ABC$. We do this by starting from D and then subtracting p, that is, $A + \overrightarrow{AB} + \overrightarrow{AC} - p$.

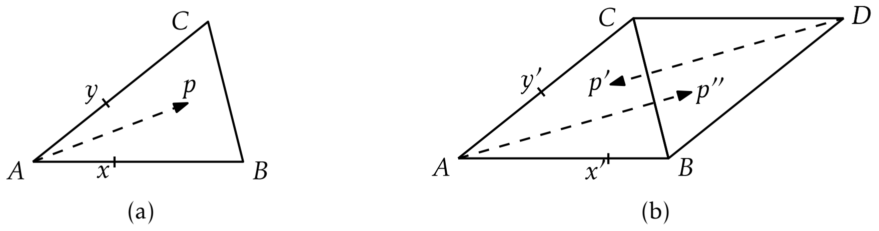

Figure 2.3: (a) A triangle $\triangle ABC$ together with a sampled point p which is obtained by computing $A + x \cdot \overrightarrow{AB} + y \cdot \overrightarrow{AC}$. (b) The parallelogram created by mirroring and flipping $\triangle ABC$, together with a sampled point p''. Since p'' lies in $\triangle BCD$, we project p'' back onto $\triangle ABC$ as p'.

Let $\mathcal{P}$ be a simple polygon. We triangulate $\mathcal{P}$ in $O(n \log n)$ time, where n is the number of vertices of $\mathcal{P}$, using the algorithm from Garey et al. [14]. Let $\mathcal{T}$ be a triangulation of $\mathcal{P}$, see Figure 2.4(a). Let *area*(x) denote the area of some geometric shape x. We compute *area*$(\mathcal{P})$ by summing up the areas of all triangles of the triangulation. For every triangle $T_i \in \mathcal{T}$ we define $prob_i$ as *area*(T_i) / *area*$(\mathcal{P})$, i.e. the proportion of the area of T_i relative to the area of $\mathcal{P}$.

Sampling points from $\mathcal{P}$ then proceeds as follows. For every $T_i \in T$ we define an interval of length $prob_i$. We shift every interval by placing them consecutively on a number of line of real numbers from 0 to 1. For example, $T_1 : [0, 0.35)$, $T_2 : [0.35, 0.75)$, $T_3 : [0.75, 1]$. We then generate an uniformly random number x from $U[0,1]$ and pick the triangle for which the shifted interval contains x. In theory, this procedure is repeated until we have a high enough probability of our sample being an ε-approximation of $\mathcal{P}$—dictated by Theorem 2.1. In practice however, we sample between 15–25 points, as this is accurate enough for our use case, as we observe in Chapter 5.

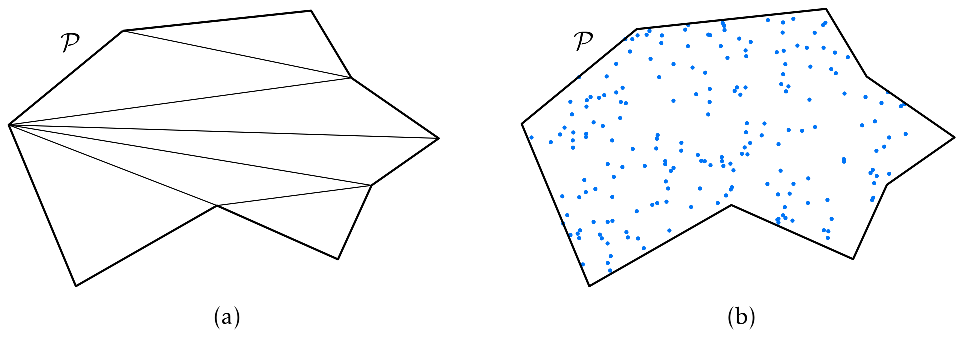

Figure 2.4: (a) A triangulation of a simple polygon $\mathcal{P}$. (b) $\mathcal{P}$ overlaid with 200 uniform random points sampled from $\mathcal{P}$.

Chapter 3

Enclosing Colored Points

In this chapter we present an algorithm on a set of colored points that computes the smallest radius disk that contains at least some number of points of a certain color, for every color. We present an algorithm that solves the problem in $O(n^2 \log n)$ time.

Given a set $P = \{p_1, ..., p_n\}$ of points in $\mathbb{R}^2$, each having a *color* between 1 and m, and a set of constraints $\{c_1, ..., c_m\}$, $c_i \in \mathbb{Z}^+$ we would like to find the smallest disk d^* with radius r^* that covers at least c_i points of color i for every i, where r^* is as small as possible. Let $color : P \to \{1, ..., m\}$ be a function that maps points to colors. We denote k as the sum of all constraints: $k = c_1 + ... + c_m$. Let $D(p,r)$ denote the disk centered around p of radius r. Let $C(p,r)$ denote the circle centered around p of radius r. We require that for every color, at least one point of that color appears in P, thus $m \leq n$. Lastly, we assume *general position*, that is, no three points lie on a straight line, and no four points lie on a circle.

We will refer to this problem as the Smallest colored k-enclosing disk problem or SCkED problem for short. To get an idea of the time complexity of the SCkED problem, we first take a look at the 1D version of the problem. We present an algorithm that solves the 1D version of the problem in $O(n \log n)$ time. Then, we present a brute force algorithm that solves the SCkED problem in $O(n^4)$ time, which shows that this problem is not NP-hard. We then present two simple 2-approximation algorithms to show that this problem can be solved with relative ease if we accept a suboptimal solution. We then formulate the SCkED problem as an LP-type problem. However, we conclude that this does not result in a running time that is at least as good as $O(n^2 \log n)$. Finally, we present an algorithm that solves the SCkED problem in $O(n^2 \log n)$ time.

3.1 Smallest Colored k Enclosing Interval

The Smallest Colored k Enclosing Interval (SCkEI) problem is similar to the SCkED, however, the points are 1-dimensional. We are interested in computing the interval with shortest length that contains at least c_i points of color i, for every i. The smallest interval is defined as an interval $I_{i,j}$, $1 \leq i \leq j \leq n$, specified by points p_i and p_j that denote the left boundary and the right boundary respectively, such that $p_j - p_i$ is as small as possible and $I_{i,j}$ contains at least c_i points of color i, for every i. The interval should enclose at least k points with several color constraints but, since we optimize for the smallest length, the actual number of points it contains does not matter. We show that this interval can be found in $O(n)$ time if the input is sorted and $O(n \log n)$ time if the input is unsorted.

Algorithm. We solve the problem by using a two pointer technique. A step-by-step visual example can be found in Figure 3.1. Using two pointers i, j we keep track of candidate intervals: intervals which contain the required number of points for every color. Of these candidate intervals we are looking for the interval of minimum length. We keep track of the counts for each color using a vector X of length m, and we keep track of how many color constraints have been

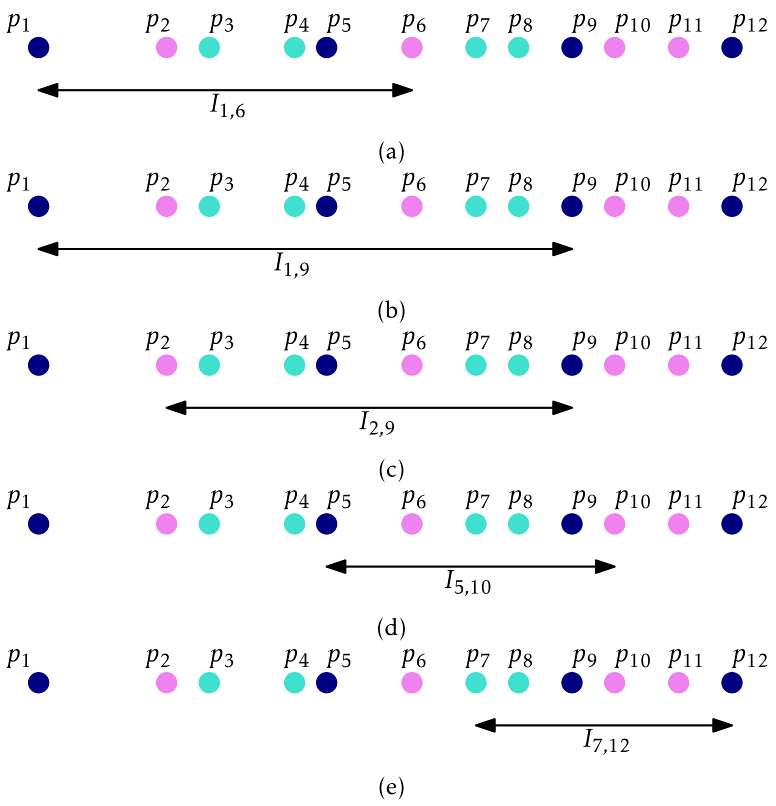
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
$I_{1,6}$
(a)
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
$I_{1,9}$
(b)
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
$I_{2,9}$
(c)
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
$I_{5,10}$
(d)
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
$I_{7,12}$
(e)

Figure 3.1: Let $P = \{p_1, \ldots, p_{12}\}$. We require that the interval contains at least two points of a certain color, for every color. (a) We first find the first interval that contains every color at least twice, $I_{1,6}$. (b) The algorithm then attempts to shrink the interval, however this is only possible whenever another blue point is added to the interval. Thus we extend the interval to $I_{1,9}$. (c) The interval shrinks to $I_{2,9}$. We can not shrink further because of the number of violet points in $I_{2,9}$. (d) The algorithm finds point p_{10} which allows the interval to shrink to $I_{5,10}$. (e) At point p_{12} we shrink the interval to $I_{7,12}$. Of the shown intervals, $I_{7,12}$ is the smallest.

fulfilled using a counter s. We use X and s as follows. Let I be an arbitrary interval. Suppose we extend I by one point p of color i. We then increment $X[i]$. If $X[i] = c_i$, then we increment s. If $s = m$, then I fulfills all color constraints. Similarly, suppose we shrink I by one point p of color i. Then we decrement $X[i]$. If $X[i] = c_i - 1$, then we decrement s. Hence, in $O(1)$ time we are able to extend and shrink an interval I by one point while keeping track of the colors present in I.

We start with $I_{i,j}$, $i = j = 1$ and increment j until $I_{1,j}$ contains a sufficient number of points such that all color constraints have been fulfilled. This is the first candidate interval and it is then stored as $I_{min} \leftarrow (1, j)$. The algorithm then enters the main loop where at every iteration one of the following events can happen.

1. If $I_{i+1,j}$ fulfills all color constraints, we set $I_{i+1,j}$ as the new candidate interval and update X and s accordingly. We then check if $I_{i+1,j}$ is smaller than I_{min}. If that is the case we set $I_{min} \leftarrow (i+1, j)$,
2. Else we set $I_{i,j+1}$ as the new candidate interval and update X and s accordingly.

The loop terminates if $j > n$.

Time complexity. The first candidate interval can be found in $O(1)$ time per step. We simply increment a counter whenever a color constraint has been fulfilled, and check whether this counter has reached m. Since $j \leq n$, this interval can be found in $O(n)$ time.

The main loop of the algorithm is executed at most $2n$ times. This is because at every iteration either i is incremented or j is incremented and we have that $1 \leq i \leq j \leq n$. Event 1 takes $O(1)$ time to execute since we only have to decrement one counter and check one color constraint. Event 2 takes $O(1)$ time to execute since we only have to increment one element of X. Hence the main loop of the algorithm consists of at most $2n$ $O(1)$ operations.

We conclude that the algorithm runs in $O(n)$ time if the input is sorted and, since we have to sort the input first, in $O(n \log n)$ time if the input is unsorted.

Correctness. In the algorithm event 1 takes precedence over event 2: We only extend the current interval to the right if it can not be made smaller from the left. Thus for every $1 \leq j \leq n$ we compute an i with $1 \leq i \leq j$ such that $I_{i,j}$ is a small as possible and fulfills all color constraints. One of these intervals must be the optimal solution and, at some point during execution, will have been stored as I_{min} in event 1.

3.2 Smallest Colored *k* Enclosing Disk

We now move to the 2D version, in which we look for the smallest disk that contains at least c_i points of color i for every i. To get a feel for the complexity of this problem, we present a brute force algorithm and show that it runs in $O(n^4)$ time. This shows that the problem is polynomial time solvable. Afterwards we look at two 2-approximation algorithms. Approximation algorithms seek to compute some result of which the value lies within some constant factor of the optimal solution. Thus a 2-approximation algorithm produces a disk of which the radius is at most twice at large as the radius of the smallest disk. We present two 2-approximation algorithms to show that this problem can be solved with relative ease if we accept a suboptimal solution.

Brute force. A smallest disk is defined by either two or three points. We divide P into $\binom{n}{3}$ subsets of size 3 and $\binom{n}{2}$ subsets of size 2. For every subset P_i with $1 \leq i \leq \binom{n}{3} + \binom{n}{2}$ we compute a disk D_i defined by the points of P_i. We then compute all points of P that lie within D_i and check if the color constrains can be fulfilled with those points. We return the disk with minimum radius that fulfills all color constraints. Computing all points that lie within some disk D_i takes $O(n)$ time. There are $\binom{n}{3} + \binom{n}{2} = O(n^3)$ of such disks. Hence, in $O(n^4)$ time we can find the disk with minimum radius that fulfills all color constraints.

2-approximation in $O(n(n/k)^2)$ time. Har-Paled et al. present a 2-approximation algorithm that approximates the radius of the smallest k-enclosing disk of a set of n points in $O(n(n/k)^2)$ time [18]. We modify their algorithm to work for a set of n colored points, while maintaining the same time complexity. Har-Paled et al. subdivide the plane into $O(n/k)$ vertical and horizontal strips that each contain at most $k/4$ points. They compute these strips in $O(n \log(n/k))$ time using median selection together with recursion. Since $O(n/k)$ horizontal lines cross with $O(n/k)$ vertical lines, there are $O((n/k)^2)$ intersections between these strips. They prove that the smallest k-enclosing disk contains at least one such intersection point. For every intersection x, they use median selection to find the kth point ordered by increasing distance from x in $O(n)$ time. Let r_x^* denote the distance from this point to x. Then, they repeat this process for every intersection x, and take the smallest r_x^*. Since the smallest k-enclosing disk contains at least one intersection point, this smallest value is a 2-approximation of r^*.

In order to modify the algorithm to work for a set of colored points, we only have to modify the step where Har-Paled et al. find the smallest k-enclosing disk centered at an intersection x. Let $P_1, ..., P_m$ be the partitioning of P by color, thus $|P_1| + ... + |P_m| = |P| = n$. For every $1 \leq i \leq m$ we apply median selection on P_i to find the c_ith point of color i, ordered by distance from x. The distance from this point to x is the radius of the disk centered at x that contains at least c_i points of color i. The total running time to compute all smallest disks centered at x is $O(|P_1|) + ... + O(|P_m|) =$

$O(n)$. Of these disks we should take the largest disk, as that is the disk that contains at least c_i points of color i, for every i. The rest of the algorithm of Har-Paled et al. is unchanged: By returning the smallest of the disks found at every intersection, we find a disk that fulfills all color constraints and has a radius that is at most twice as large as the optimal radius.

2-approximation in $O(n^2)$ time. We iterate over all points of P. For every point $p \in P$ we compute the smallest disk centered at p that fulfills all color constraints. Computing this smallest disk can be done in $O(n)$ time in the same manner as described in the previous algorithm. We then take the smallest of these disks. The smallest disk must contain at least one point of P. In the best case the center of the smallest disk is located exactly at some point of P. In that case this algorithm will find the optimal solution. In the worst case all points of P lie on the boundary of the smallest disk. In this case the smallest disk found has a radius that is twice the size of the optimal radius. We spend $O(n)$ time per point, hence this algorithm takes $O(n^2)$ time.

3.3 LP-type

In this section we explore the possibilities of using the abstract framework for solving LP-type problems by Sharir and Welzl [30]. However, in order for this approach to be feasible, we should be able to solve the SCkED problem for a set of points of at most four colors in $O(n^2)$ time. We currently do not see a way to achieve this running time. Hence, a faster algorithm can be found in Section 3.4. Nonetheless, it is interesting to look at the possibilities of interpreting the SCkED problem as an LP-type problem.

LP-type problems have as input a finite set of points P called constraints, and a function $f : 2^P \to F$ where 2^P is the powerset of P and F is a totally ordered set of values. This function is required to satisfy two axioms:

1. For every two sets A, B:
$$A \subseteq B \subseteq P \Longrightarrow f(A) \leq f(B)$$
2. For every two sets A, B and every element $p \in P$:
$$A \subseteq B \subseteq P,\ p \in P,\ f(A) = f(B) < f(A \cup \{p\}) \Longrightarrow f(B) < f(B \cup \{p\})$$

A typical LP-type problem is the smallest enclosing circle problem. Let P be a set of points in $\mathbb{R}^2$. The objective is to find the smallest circle that encloses all points. Let $f(P)$ be the radius of the smallest enclosing circle of P. We show that the two axioms hold for this problem. Let $A \subseteq B \subseteq P$ be two subsets of P. The first axiom is satisfied because clearly the smallest enclosing circle of A can not be larger than the smallest enclosing circle of B, thus $f(A) \leq f(B)$. Let $p \in P$ be an arbitrary point and let $f(A) = f(B)$. Since $f(A) < f(A \cup \{p\})$, point p lies outside of the smallest enclosing circle of A. Since $f(A) = f(B)$ and $A \subseteq B$, the smallest enclosing circles of A and B must be identical. Hence we have that p also lies outside of the smallest enclosing circle of B thus $f(B) < f(B \cup \{p\})$.

In LP-type problems a *basis* $\mathcal{B}$ is defined as a set of constraints where, for all $\mathcal{B}' \subset \mathcal{B}$, we have that $f(\mathcal{B}') < f(\mathcal{B})$ [30]. The *combinatorial dimension* d is the maximum cardinality of a basis. In LP-type problems we start off with an initial base $\mathcal{B}_0$ and build from there. A constraint $h \in P$ *violates* basis $\mathcal{B}$ if $f(\mathcal{B} \cup \{h\}) > f(\mathcal{B})$, which implies that $h \notin \mathcal{B}$.

We specify two primitive operations dictated by the LP-type framework:

- Violation test – Given a basis $\mathcal{B}$ and constraint $h \in P$ is it true that $f(\mathcal{B} \cup \{h\}) > f(\mathcal{B})$?
- Basis construction – Given a basis $\mathcal{B}$ and constraint $h \in P$ compute some basis of $\mathcal{B} \cup \{h\}$.

For the smallest enclosing circle problem, a basis contains at most three points. Thus we perform the violation test in $O(1)$ time simply by testing if a point h lies within the circle defined by the at most three points of $\mathcal{B}$. We construct a basis $\mathcal{B}$ of a set of at most four points by searching for its smallest enclosing circle. We can do this by testing all combinations of two or three points. of which there are at most $\binom{4}{3}+\binom{4}{2}=10$. Thus we perform the basis construction operation in $O(1)$ time.

Randomized algorithms are known that solve LP-type problems in $O(dmT+t(d)E\log m)$ time. Where T is the running time of the violation test, E is the running time of the basis construction, $t(d)$ is exponential or subexponential in d, and m is the number of constraints [22, 17]. Using this algorithm we solve the smallest enclosing disk problem for a set of m points in expected $O(m)$ time.

Point constraints. To solve the SCkED problem, a logical strategy would be to color the points and reuse the proofs of the axioms for the smallest enclosing disk problem. However, we show that this approach does not work. Let the input P be a set of colored points and let $f(P)$ be the radius of the SCkED of P. In Figure 3.2 we show that the first axiom does not hold. Let A and B be sets containing points of two colors. As shown in Figure 3.2, we can build sets A and B such that $A \subseteq B$ and $f(A) > f(B)$, violating the first axiom.

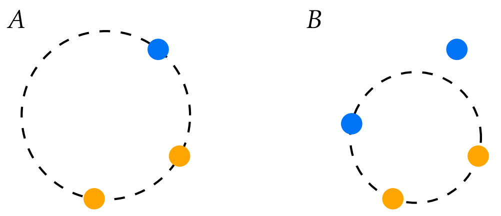

Figure 3.2: In this figure A and B are sets of colored points. We would like to find the smallest disk that contains at least two orange points and at least one blue point. We have that $A \subseteq B$, however clearly $f(A) > f(B)$.

Constraints as color classes. An approach that does satisfy the axioms is to treat color classes of points as constraints. For a color class P_i of points of color i, we say that P_i is covered by some disk d if at least c_i points of P_i lie within d. Let $\mathcal{P}$ be the set of color classes and let $f(\mathcal{P})$ be the radius of the smallest disk that covers every color class in $\mathcal{P}$. We prove that the axioms hold for this combination of input set and function.

Proof. Let $\mathcal{F}$ and $\mathcal{G}$ be two sets of classes of points. Since $\mathcal{F} \subseteq \mathcal{G}$, any disk that is valid for $\mathcal{G}$, is also valid for $\mathcal{F}$. Thus $f(\mathcal{F}) \le f(\mathcal{G})$, satisfying the first axiom.

In order to prove the second axiom let $H \in \mathcal{P}$ be a color class, let $\mathcal{F}$ and $\mathcal{G}$ be two sets of classes of points with $\mathcal{F} \subseteq \mathcal{G}$. Suppose that $f(\mathcal{G}) = f(\mathcal{F}) < f(\mathcal{F} \cup \{H\})$. Then H must be a color class that lies (partly) outside of the SCkED of $\mathcal{F}$. Since $\mathcal{F} \subseteq \mathcal{G}$ and $f(\mathcal{G}) = f(\mathcal{F})$ the SCkED of $\mathcal{F}$ and $\mathcal{G}$ must be identical. Hence we have that H also lies (partly) outside of the SCkED of $\mathcal{G}$ thus $f(\mathcal{G}) < f(\mathcal{G} \cup \{H\})$. □

The SCkED is always defined by at most three different colors. Thus if we have a set of four or more color classes, at least one of the classes will not partake in determining the size of the SCkED. Hence, for this problem the combinatorial dimension is three. Constructing the initial base is a problem equivalent to the SMALLEST k-ENCLOSING DISK problem. This problem can be solved in $O(nk)$ expected time [18].

The violation test can be formulated as follows: Given three color classes that together determine a disk $D(\mathcal{B})$, do at least c_H of the points of H lie within $D(\mathcal{B})$? If that is not the case then $D(\mathcal{B})$ has to be expanded meaning $f(\mathcal{B} \cup \{H\}) > f(\mathcal{B})$. The total number of points contained in H and the color classes of $\mathcal{B}$ is at most n. If we are given $D(\mathcal{B})$ then we iterate over all points in H and count the points that lie within $D(\mathcal{B})$. Hence, in at most $|H| = O(n)$ steps we perform the violation test.

As part of the basis construction operation, we have to compute the SCkED of a set of points of at most four colors. Recall that it takes $O(dmT + t(d)E \log m)$ to solve an LP-type problem, where m translates to the number of colors. In Section 3.4 we present an algorithm that solves the SCkED problem in expected $O(n^2 \log n)$ time. Hence, in order to achieve a better running time of, say, $O(n^2 \log m)$, the basis construction should run in $O(n^2)$ time. However, the constant (four) number of colors does not help us here. We do not see a way to solve the SCkED problem for a set of points of at most four colors in $O(n^2)$ time. In the next section we present an algorithm that solves the SCkED problem in $O(n^2 \log n)$ expected time. We could use this as a basis construction algorithm, but this would result in a running time of $O(n^2 \log n \log m)$. It would be faster to directly run the $O(n^2 \log n)$ algorithm on the input.

3.4 A Sweep Algorithm

We draw inspiration from the algorithms by Efrat et al. [12] and Matoušek [21] in order to build an efficient algorithm that solves the Smallest Colored k-Enclosing Disk problem. In this section we present a randomized algorithm that runs in $O(n^2 \log n)$ expected time. The general idea of this algorithm is to compute, for a point $p \in P$, the smallest colored k-enclosing disk of which p lies on the boundary. We denote the radius of this disk as r_p^*. The key idea is that we are able to test in $O(n \log n)$ time and for an arbitrary radius r, if $r < r_p^*$. This allows us to find r_p^* in $O(n^2)$ time. We then use the same test in a randomized approach to find the smallest r_q^*, for $q \in P$ in $O(n^2 \log n)$ time.

3.4.1 Definitions

For some $p \in P$ and some radius r, $I(p,r) \subseteq P$ denotes the set of points of P that lie within a distance of $2r$ from p. Let d_p^* be the smallest disk that fulfills all color constraints, for which p lies on the boundary. Let r_p^* be its radius. Let $r_0 \geq r^*$ be some upperbound of r^*. Let *cdepth*(p,r) (which stands for color depth) of some point p with radius r be a vector of length m that contains for every color i the number of points of color i that lie in $D(p,r)$. We say that a color depth fulfills the color constraints if *cdepth*$(p,r)[i] \geq c_i$ for $1 \leq i \leq m$. Using the definition of r_p^* Matoušek [21] redefines r^* as

$$r^* = \min_{p \in P} r_p^*$$

We have that $|I(p,r_0)| = O(n)$ for $r_0 \geq r^*$. The reason why the algorithm of Matoušek [21] is able to run in $O(n \log n + nk)$ time is because the author is able to find a value $r_0 \geq r^*$ such that $|I(p,r_0)| = O(k)$. We argue that such an r_0 might not exist in our problem setting.

Let G_r the subdivision of the plane into square cells of width r. Let *depth*(r) be the number of points a disk of radius r can contain and let *gdepth*(r) be the maximum number of points contained in a single grid cell of G_r. Matoušek [21] proves the following lemma:

Lemma 3.1 ([21]). *gdepth*$(r) \leq$ *depth*$(r) = O($*gdepth*$(r))$

Matoušek presents an algorithm that finds a value $r_0 \geq r^*$ such that *gdepth*$(r_0) = k$ and $k \leq$ *depth*$(r_0) = O(k)$ [21]. By giving a counterexample, we show that such an r_0 might not exist in our problem setting.

Suppose $P = \{p_1, \dots, p_n\}$ and $color(p_1) = 1$, $color(p_2) = 2$, $color(p_i) = 3$ for $3 \leq i \leq n$. We would like to cover at least one point of all three colors, thus $k = 3$. Suppose that the boundary of the SCkED passes through two points: p_1 and p_2, as illustrated in Figure 3.3. We define two grid cells A and B of width r^*. The remaining points, $p_3, \dots, p_n$, lie in cell A. Cell A contains $n-1$ points which, by Lemma 3.1, implies that $n - 1 \leq$ *depth*$(r^*) = O(n)$. We conclude that any disk of radius $r_0 \geq r^*$ may contain $O(n)$ points, thus $|I(p,r_0)| = O(n)$.

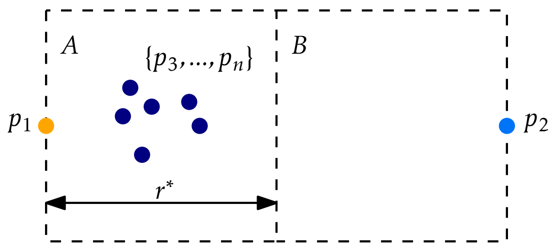

Figure 3.3: Two cells of G_{r^*}. We would like to cover at least one point of every color. In this setting $k = 3$, grid cell A contains $n-1$ points, thus $gdepth(r^*) = n-1$.

3.4.2 Testing whether $r < r_p^*$

Matoušek [21] introduces a test that determines whether $r < r_p^*$, for a given r and for any $p \in P$. In this section we modify this test to work for a set P of colored points. This test is also illustrated in Figure 3.4(a). In this figure arcs of $C(p,r)$ are labeled with the corresponding color depth.

We compute the intersections of $C(q,r)$ with $C(p,r)$ for all $q \in I(p,r)$. There are $O(n)$ of such intersections. We sort these intersections by their clockwise order on $C(p,r)$. Afterwards, we compute the color depth of the rightmost point of $C(p,r)$. We then perform a walk along the boundary of $C(p,r)$, updating the color depth at every intersection. At every arc of $C(p,r)$, a disk of radius r can be constructed centered somewhere on that arc, that contains colors described by the color depth of the arc. As an example, in Figure 3.4(a) a disk d of radius r is shown that is centered on an arc of $C(p,r)$ that is labeled by $[2,0]$. This means that two points of color 1 and no points of color 2 lie within d. If at no arc the color depth fulfills all color constraints, then at no point we are able to construct a disk that fulfills all color constraints. We then conclude that $r < r_p^*$, otherwise we conclude that $r \geq r_p^*$.

We sort the intersections in $O(n \log n)$ time. Computing the color depth of the rightmost point is done in $O(n)$ time by checking in which circles the rightmost point lies. We maintain a counter that keeps track of how many color constraints are fulfilled. If a color constraint becomes fulfilled at a certain intersection, then the counter is incremented. If a color constraint becomes unfulfilled, the counter is decremented. Instead of checking if all color constraints are fulfilled, we check if the counter has reached a value of m. At every intersection one color is updated. Note that intersections may overlap, for example when three circles have a common intersection point. We handle overlapping intersections seperately. Thus the total running time of this test is $O(n \log n)$.

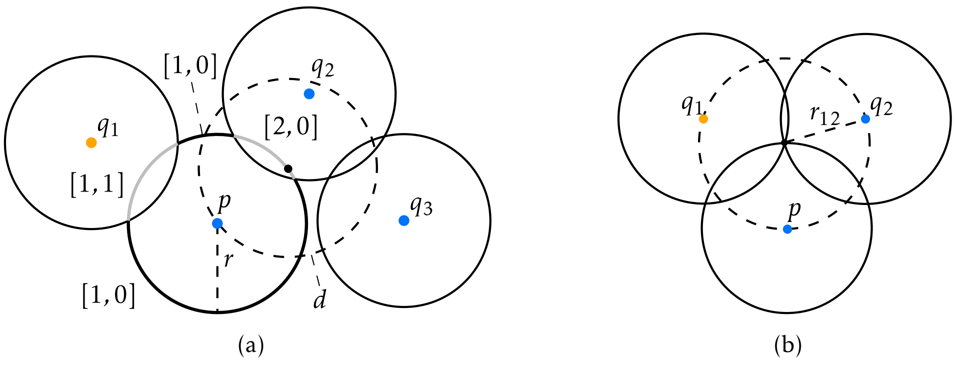

Figure 3.4: $P = \{p, q_1, q_2, q_3\}$. We want to include at least two blue points and at least one orange point. (a) Arcs on $C(p,r)$ are visualized by alternating black and gray curves. At no point in the walk along the boundary of $C(p,r)$ are all color constraints fulfilled. Hence we conclude that $r < r_p^*$. (b) $C(p,r)$, $C(q_1,r)$, $C(q_2,r)$ share an intersection point.

3.4.3 Computing r_p^*

We use the previous test in an algorithm inspired by the technique of *parametric search*. Parametric search is a technique that allows us to transform a decision algorithm into an optimization algorithm. The technique was invented by Nimrod Megiddo [23]. We require a decision problem $\mathcal{P}(r)$ that is monotone in r, that is, if $\mathcal{P}(r_1)$ is true, then $\mathcal{P}(r_2)$ is true, for all $r_2 \geq r_1$. In our case this decision problem is: Given a radius r, is there disk of radius r with p on its boundary, that fulfills all color constraints? We are looking for the smallest r that allows for the existence of such a disk.

For a given r, there is a radius $r' < r$ such that, for all values $\overline{r}$ in $[r', r)$ the system of arcs on $C(p, \overline{r})$ remains the same. This radius r' is what we call a *critical radius*, that is, a radius where the system of arcs on $C(p, r')$ induced by $C(q, r')$, $q \in I(p, r)$ changes combinatorially. For a radius r, the system of arcs changes combinatorially whenever $C(p, r)$ touches another circle $C(q, r)$, or whenever $C(p, r)$ has a common intersection point with two other circles $C(q_i, r)$, $C(q_j, r)$. We prove the following lemma, which allows us to search for r_p^* among the critical radii.

Lemma 3.2. *r_p^* is a critical radius.*

Proof. Assume that r_p^* is not a critical radius. Then there must be a critical radius $r' < r_p^*$ for which the system of arcs on $C(p, r')$ is the same as the system of arcs on $C(p, r_p^*)$. This implies that there exists a disk of radius r' that has p on its boundary and satisfies all color constraints. Thus, by definition, $r' \geq r_p^*$. This is a contradiction, hence r_p^* must be a critical radius. □

Let $A = \{dist(p, q) \ / \ 2 \ \mid q \in P\}$ be the set of radii for which two circles touch. Let B be the set of radii for which three circles have a common intersection point. We build B is as follows. For three points p, q_i, q_j, and a radius r, circles $C(p, r)$, $C(q_i, r)$, and $C(q_j, r)$ have a common intersection point when there exists a point that has the same distance to all three circles. Let this distance be r_{ij}. This distance is equal to the radius of the circle defined by p, q_i, q_j (Figure 3.4(b)). We can now define set B as the set of all radii of circles defined by points p, q_i, q_j, for all q_i, q_j. Let this set of critical radii be denoted by $R_c = A \cup B$. This set has cardinality $O(n^2)$, dominated by the size of B.

Let r' be the median value of R_c. If $r' < r_p^*$, then we discard all $r \in R_c$ for which $r \leq r'$. Similarly, if $r' \geq r_p^*$, then we discard all $r \in R_c$ for which $r > r'$. We repeat this process until there is a single radius left in R_c. This final radius is r_p^*, due to Lemma 3.2.

Median selection takes linear time with relation to the number of elements in the set. Since we discard half of the radii after every median selection, the running time of finding r_p^* is

$$\sum_{i=0}^{\log n} \frac{1}{2^i} \cdot O(n^2) = O(n^2)$$

This brings us to the following lemma:

Lemma 3.3. *For some $p \in P$, we can compute r_p^* in $O(n^2)$ time.*

3.4.4 Computing r^*

Combining Lemma 3.3 with the definition of r^* results in a simple $O(n^3)$ algorithm for computing r^*, by simply computing r_p^* for every $p \in P$ and taking the smallest value. Using the *prune and search* technique, we reduce this running time to $O(n^2 \log n)$. The prune and search technique has been introduced by Megiddo [24]. The basic idea consists of a recursive algorithm in which the search space is repeatedly reduced by pruning a fraction of the input set at every recursion. We can implement the pruning step as follows. Let Q be a copy of P. We select a random point $p \in Q$ and compute r_p^* using the points of P. We then discard all points q from Q for which $r_q^* \geq r_p^*$, also discarding p itself. We then recurse with P and the remainder of Q. If Q is empty, then r_p^* was the optimal solution. The reason why we work with two sets is that we should maintain a complete set of points to compute r_p^* and to perform the test from Section 3.4.2. In pseudocode this routine looks as follows.

Algorithm 1 This algorithm computes the smallest colored k-enclosing disk of a set of points P. Function RandomElement returns a random element from the input set. The initial call of this algorithm is Find $r^*(P,P)$, with $n = |P|$.

```
1:  procedure Find $r^*(P,Q)$
2:      $p \leftarrow$ RandomElement($Q$)
3:      Compute $r_p^*$                                    ▷ $O(n^2)$
4:      for $q \in Q$ do                                  ▷ $O(|Q| n \log n)$
5:          Delete $q$ from $Q$ if $r_q^* \geq r_p^*$     ▷ Using the test from Section 3.4.2
6:      end for
7:      if $Q = \emptyset$ then
8:          return $r_p^*$
9:      else
10:         return Find $r^*(P,Q)$                        ▷ $O(|Q| n \log n) + O(n^2)$
11:     end if
12: end procedure
```

Theorem 3.4. *The* smallest colored k-enclosing disk *can be computed in expected $O(n^2 \log n)$ time, for a set P of n colored points.*

Proof. The total running time of this algorithm can be described using a function $T(n,m)$, where n represents the size of P and m represents the size of Q (Algorithm 1). Initially $m = n$, thus the total running time of the algorithm is described by $T(n,n)$.

If $m = 1$, we do not recurse and thus the running time is dominated by the running time required to compute r_p^*, which is $O(n^2)$. Otherwise, we recurse on the reduced set Q with cardinality at least 1 and at most $m-1$, all with an equal probability of $\frac{1}{m-1}$. Let X_k be an indicator random variable, which is 1 if the reduced set Q contains k elements, and 0 otherwise. Therefore, the expected running time of Algorithm 1 as

$$\mathbf{E}[T(n,m)] = \begin{cases} O(n^2) & \text{if } m = 1 \\ O(mn\log n) + O(n^2) + \mathbf{E}[\sum_{k=1}^{m-1} T(n,k)\cdot X_k] & \text{if } m > 1 \end{cases}$$

We can further reduce this expression as follows using the *linearity of expectation*: The expected value of a sum of random variables is equal to the the individual expected values.

$$\begin{aligned} &\mathbf{E}[\sum_{k=1}^{m-1} T(n,k)\cdot X_k] \\ &= \sum_{k=1}^{m-1} \mathbf{E}[T(n,k)\cdot X_k] && \{\text{ Linearity of expectation }\} \\ &= \sum_{k=1}^{m-1} \mathbf{E}[T(n,k)]\cdot \mathbf{E}[X_k] && \{\ T(n,k) \text{ and } X_k \text{ are independent }\} \end{aligned}$$

The value of $\mathbf{E}[X_k]$ is the probability that $X_k = 1$. Since the reduced set Q has cardinality at least 1 and at most $m-1$, all with an equal probability, $\mathbf{E}[X_k] = \frac{1}{m-1}$. This allows us to rewrite the recurrence as follows.

$$\mathbf{E}[T(n,m)] = \begin{cases} O(n^2) & \text{if } m = 1 \\ O(mn\log n) + O(n^2) + \frac{1}{m-1}\sum_{k=1}^{m-1} \mathbf{E}[T(n,k)] & \text{if } m > 1 \end{cases}$$

We show that $\mathbf{E}[T(n,m)] = O(mn\log n + n^2\log m)$, for $m \geq 2$, by proving that

$$\mathbf{E}[T(n,m)] \leq Amn\log n + Bn^2\max(1,\log m)$$

for an appropriate choice of positive constants A and B. We prove this for all $m \geq 2$ using induction on m.

Base case ($m = 1$): By definition of T we have $\mathbf{E}[T(n,1)] \leq Cn^2$, for some positive constant C. For $m = 1$, we should show that $Cn^2 \leq An\log n + Bn^2$. We pick $A > 0$ and $B \geq C$ to satisfy this inequality. Hence for $A > 0$, $B \geq C$, and $m = 1$ we have that

$$\mathbf{E}[T(n,1)] \leq Amn\log n + Bn^2\max(1,\log m)$$

Base case ($m = 2$): By definition of T we have

$$\begin{aligned}
&\mathbf{E}[T(n,2)]\\
&\leq Cmn\log n + Dn^2 + \frac{1}{m-1}\sum_{k=1}^{m-1}\mathbf{E}[T(n,k)] && \{\text{ Def of T, for positive constants } C \text{ and } D\ \}\\
&= Cmn\log n + Dn^2 + \mathbf{E}[T(n,1)]\\
&\leq Cmn\log n + Dn^2 + En^2 && \{\text{ Def. of T, for positive constant } E\ \}
\end{aligned}$$

Hence for $A \geq C$, $B \geq \frac{D+E}{\log 2}$, and $m = 2$ we have that

$$\mathbf{E}[T(n,2] \leq Amn\log n + Bn^2\max(1,\log m)$$

Induction hypothesis: $\mathbf{E}[T(n,m)] \leq Amn\log n + Bn^2\max(1,\log m)$ for positive constants A, B.

Step ($m+1$, $m \geq 2$):

$$\begin{aligned}
&\mathbf{E}[T(n,m+1)]\\
&\leq C(m+1)n\log n + Dn^2 + \frac{1}{m}\sum_{k=1}^{m}\mathbf{E}[T(n,k)] && \{\text{ Def. of } T,\ C,D > 0\ \}\\
&\leq C(m+1)n\log n + Dn^2 + \frac{1}{m}\sum_{k=1}^{m}(Akn\log n + Bn^2\max(1,\log k)) && \{\text{ IH }\}\\
&= C(m+1)n\log n + Dn^2 + \frac{An\log n}{m}\sum_{k=1}^{m}k + \frac{Bn^2}{m}\sum_{k=1}^{m}\max(1,\log k)\\
&= C(m+1)n\log n + Dn^2 + \frac{A(m+1)n\log n}{2} + \frac{Bn^2}{m}\sum_{k=1}^{m}\max(1,\log k) && \{\ \sum_{k=1}^{m}k = \frac{m(m+1)}{2}\ \}\\
&= C(m+1)n\log n + Dn^2 + \frac{A(m+1)n\log n}{2} + \frac{Bn^2}{m}\left(1 + \sum_{k=1}^{m}\log k\right)\\
&= C(m+1)n\log n + Dn^2 + \frac{A(m+1)n\log n}{2} + \frac{Bn^2}{m} + \frac{Bn^2}{m}\sum_{k=1}^{m}\log k
\end{aligned}$$

We compute the appropriate value for A as follows.

$$\begin{aligned}
&C(m+1)n\log n + \frac{A(m+1)n\log n}{2} \le A(m+1)n\log n \\
&\Leftrightarrow C(m+1)n\log n \le \frac{A(m+1)n\log n}{2} \\
&\Leftrightarrow C(m+1) \le \frac{A(m+1)}{2} \\
&\Leftrightarrow C \le \frac{A}{2}
\end{aligned}$$

We then compute the appropriate value for B as follows.

$$\begin{aligned}
&Dn^2 + \frac{Bn^2}{m} + \frac{Bn^2}{m}\sum_{k=1}^{m}\log k \\
&\le Dn^2 + \frac{Bn^2}{m} + \frac{Bn^2}{m}\left(\left\lceil\frac{m}{2}\right\rceil\log m + \left\lfloor\frac{m}{2}\right\rfloor\log\frac{m}{2}\right) \\
&= Dn^2 + \frac{Bn^2}{m} + \frac{Bn^2}{m}\left(\left\lceil\frac{m}{2}\right\rceil\log m + \left\lfloor\frac{m}{2}\right\rfloor\log m - \left\lfloor\frac{m}{2}\right\rfloor\log 2\right) \\
&= Dn^2 + \frac{Bn^2}{m} + \frac{Bn^2}{m}\left(m\log m - \left\lfloor\frac{m}{2}\right\rfloor\right) \\
&= Dn^2 + \frac{Bn^2}{m} + Bn^2\log m - \left\lfloor\frac{m}{2}\right\rfloor\frac{Bn^2}{m} \\
&\le Dn^2 + \frac{Bn^2}{m} + Bn^2\log m - \frac{Bn^2}{3} && \left\{ \left\lfloor\frac{m}{2}\right\rfloor/m \ge \frac{1}{3} \text{ for } m \ge 2 \right\}
\end{aligned}$$

$$\begin{aligned}
&Dn^2 + \frac{Bn^2}{m} + Bn^2\log m - \frac{Bn^2}{3} \le Bn^2\log(m+1) \\
&\Leftrightarrow D + \frac{B}{m} + B\log m - \frac{B}{3} \le B\log(m+1) \\
&\Leftrightarrow D \le B\log(m+1) - \frac{B}{m} - B\log m + \frac{B}{3} \\
&\Leftrightarrow D \le B(\log(m+1) - \frac{1}{m} - \log m + \frac{1}{3}) \\
&\Leftarrow D \le \frac{B}{3} && \left\{ \log(m+1) \ge \log m + \frac{1}{m} \text{ for } m \ge 1 \right\}
\end{aligned}$$

Hence, for $A \ge 2C$, $B \ge 3D$, and for all $m \ge 3$ we have that

$$\mathbf{E}[T(n,m)] = Cmn\log n + Dn^2 + \frac{1}{m-1}\sum_{k=1}^{m-1}\mathbf{E}[T(n,k)] \le Amn\log n + Bn^2\max(1,\log m)$$

We combine this with the base cases to get $\mathbf{E}[T(n,m)] = O(mn\log n) + O(n^2\log m)$ for all $m \ge 1$. In our algorithm, $m = n$ initially, and hence we conclude that the expected total running time of the algorithm is

$$\mathbf{E}[T(n,n)] = O(n^2\log n)$$

□

Chapter 4

Star-Shaped Necklaces

As we have discussed in Chapter 1, necklaces do not always have to be circles. A necklace that follows the shape of the map more closely, is often preferred. An example is the continent of Africa, see Figure 4.1: it would look better if, instead of a circle, we draw a star-shaped necklace around the continent. For this we should take the global shape of the map into account. In this chapter we first model the global shape as a set of *feature points*—points that describe the important features of the continent. Then, we draw Bézier curves through every pair of consecutive feature points in order to build the necklace. Finally, we show how we can represent a circle using Bézier curves and use that to interpolate between circular necklaces and star-shaped necklaces. We allow interpolation between circular necklaces and star-shaped necklaces to improve flexibility. As we will see in Chapter 5, the most suitable is usually a star-shaped curve that has been slightly interpolated towards a circle.

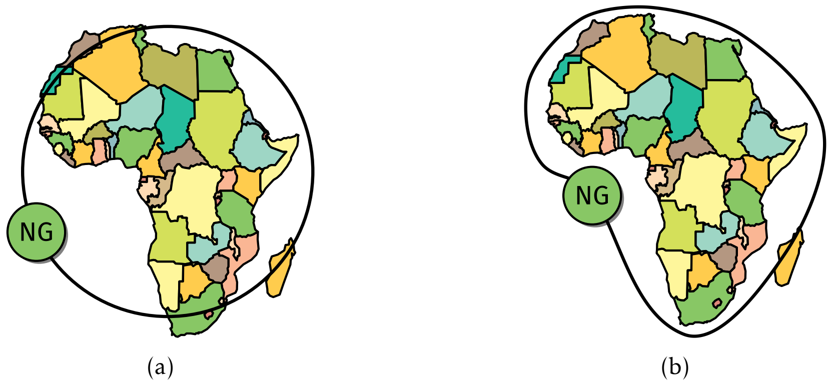

Figure 4.1: (a) A circular necklace is shown which leaves a significant amount of empty space in the bottom left quadrant of the necklace. (b) A star-shaped necklace, which is better suited for this map.

4.1 Computing the Feature Points

In order to compute a star-shaped necklace for a set of regions, which are represented by polygons $P_1, ..., P_m$, we first compute the union of the regions. We do this by recursively combining polygons that share a border, into a single polygon. In the end we obtain a set of polygons that do not share a border—think of mainland Africa and Madagascar as two separate polygons. Our main goal is to capture the shape of the polygon. Since a star-shaped necklace should only roughly follow the continent it surrounds, it suffices to model the global shape as a small set of points derived from the *medial axis*, see Figure 4.2. The medial axis was introduced by Blum [6]

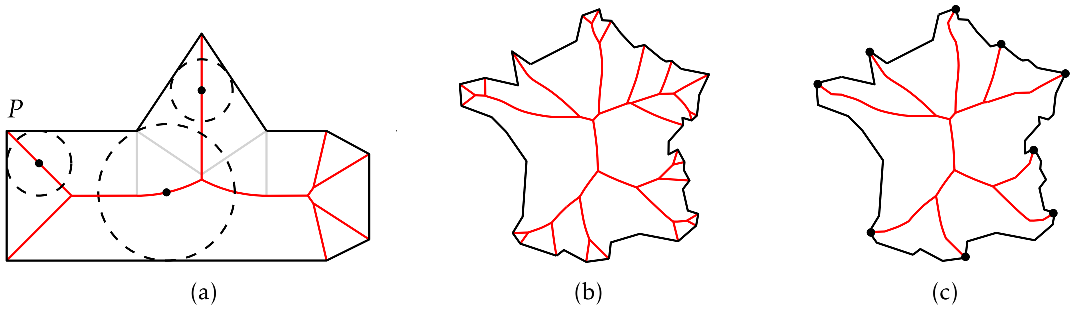

Figure 4.2: (a) A concave polygon with its medial axis shown in red. Every point on the medial axis is the center of some circle that touches the boundary of the polygon in at least two places. The gray edges would be in the Voronoi diagram of line segments, but are not part of the medial axis. (b) The medial axis of France. (c) A pruned medial axis of France together with the corresponding feature points.

in the field of biology, who introduced it as the *topological skeleton*. The medial axis of polygon P is defined as the set of points with more than one closest point on the boundary of P. Intuitively, one can draw a circle centered on every point of the medial axis that touches the boundary of the polygon in at least two places. The medial axis is related to the Voronoi diagram. In fact, the medial axis of a polygon P is a subset of the *Voronoi diagram of line segments* of P. The Voronoi diagram of line segments is a partition of the plane into cells such that every point in a cell has the same nearest line segment. For a simple polygon the medial axis can be computed in linear time [8]. For convex polygons, the medial axis consists of just straight line segments, whereas for concave polygons the medial axis contains curved line segments.

The medial axis M of a polygon P is a tree whose leaves coincide with vertices of P. We use the leaves of M as the set of feature points. When the boundary of P is noisy, M can get very large. In order to capture only the most important features of the polygon, we remove branches from the tree which do not contribute as much to the shape of the continent as other branches. The process of removing branches from a tree is called *pruning*. Various methods for pruning the medial axis have been explored in the past [29]. Bai and Letecki [3] present a method for pruning the medial axis that optimizes for shape reconstruction, that is, their pruning method tries to conserve the shape of the polygon as well as possible. Because our main goal is modeling the shape of the polygon, we use their method to prune the medial axis. Bai and Letecki iteratively remove the *end-branch* with the smallest relevance for shape reconstruction, see Figure 4.3. An end-branch b is a branch between a leaf of the medial axis and its closest parent, denoted by $parent(b)$. The *maximum inscribed circle* of a node in M is the largest circle centered on that that node that lies completely within P. The end-branch with the smallest relevance for shape reconstruction is the end-branch that contributes least to the area of P. The area that an end-branch b contributes is the area internal of P bounded by the maximum inscribed circle of $parent(b)$.

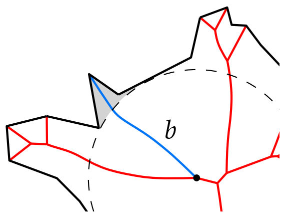

Figure 4.3: An end-branch b shown in blue. The black point represents $parent(b)$. The gray shaded area is the area b contributes.

In Section 4.2 we will draw a Bézier spline through the feature points. In Chapter 1 we discussed how the star-shaped necklace should be a smooth shape. In order to achieve additional visual smoothness, we require the Bézier curves to be similar in length. The curves should also not be too short. Since we draw Bézier curves through the feature points, the length of the Bézier curves is tied to the number of feature points, which in turn is tied to the number of end-branches in the medial axis. Thus we should repeatedly prune end-branches until there are only a few feature points left. In order to enforce these constraints we define a user-defined parameter $\gamma \geq 0$. We want the distance between any pair of consecutive feature points to be at least γ. After every pruning step, we compute the minimum distance between any pair of consecutive feature points. If this distance is larger than γ, we perform another pruning step. We repeat this process until the distances between every pair of consecutive feature points are at least γ.

4.2 The Necklace

Bézier curves are parametric equations that were originally introduced by Paul de Casteljau [9]. They are defined by a set of control points. A Bézier curve defined by two control points is called a linear Bézier curve and is used to draw straight lines. In Figure 4.4 quadratic and cubic curves are shown. A quadratic Bézier curve is defined by three control points and is used to draw shapes that look like parabola. A cubic Bézier is defined by four control points and may contain turns. Since our necklace may require turns, we use cubic Bézier curves. The general idea is that we chain Bézier curves together trough the feature points to form a single star-shaped C^1-continuous curve. This curve can be open or closed depending on whether or not we connect the first and the last Bézier curve using another Bézier curve. Various levels of continuity can be enforced. A Bézier spline is always at least C^0-continuous, that is, the spline contains no holes. A Bézier spline is C^1-continuous (smooth) when its derivative is continuous. Two Bézier curves form a C^1-continuous spline when they are connected and the third control point of the first curve lines up with the second control point of the second curve—see Figure 4.4(b).

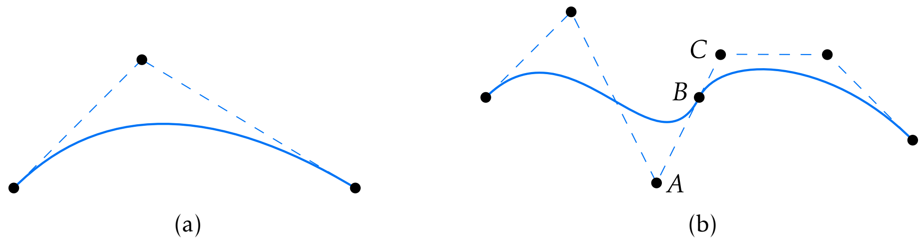

Figure 4.4: (a) A quadratic curve. (b) Two cubic curves joined through point B. Because A and C lie on a straight line through B, this curve is C^1-continuous.

We use the feature points described in the previous section as start and endpoints for Bézier curves. We will now formally define the necklace in terms of Bézier curves. The following explanation has also been visualized in Figure 4.5(a) and Figure 4.5(b). Let n be the number of feature points. For every pair of two consecutive feature points F_i, F_{i+1}, with $1 \leq i \leq n$ and $n+1 \equiv 1$, we create Bézier curves C_i for all i and set F_i and F_{i+1} as the first and fourth control points of C_i respectively. For a Bézier curve C_i let $P_{i,j}$ with $1 \leq j \leq 4$, denote the jth control point of curve C_i. The control points are ordered along the spline in counterclockwise direction. We have that $P_{i,4} = F_{i+1} = P_{i+1,1}$. To make two Bézier curves, C_i, C_{i+1}, C^1-continuous at point F_{i+1} we place $P_{i,3}$ and $P_{i+1,2}$ on a straight line through F_{i+1}.

We should specify the distance between $P_{i,3}$ and F_{i+1}, and between $P_{i+1,2}$ and F_{i+1}. We choose to make this distance dependent on the distance between the feature points multiplied by a factor. Experiments show that 0.25 is a good value for this factor. We set the the distance between $P_{i,3}$ and F_{i+1} to $0.25 \cdot d(F_i, F_{i+1})$, and the distance between $P_{i+1,2}$ and F_{i+1} to $0.25 \cdot d(F_{i+1}, F_{i+2})$.

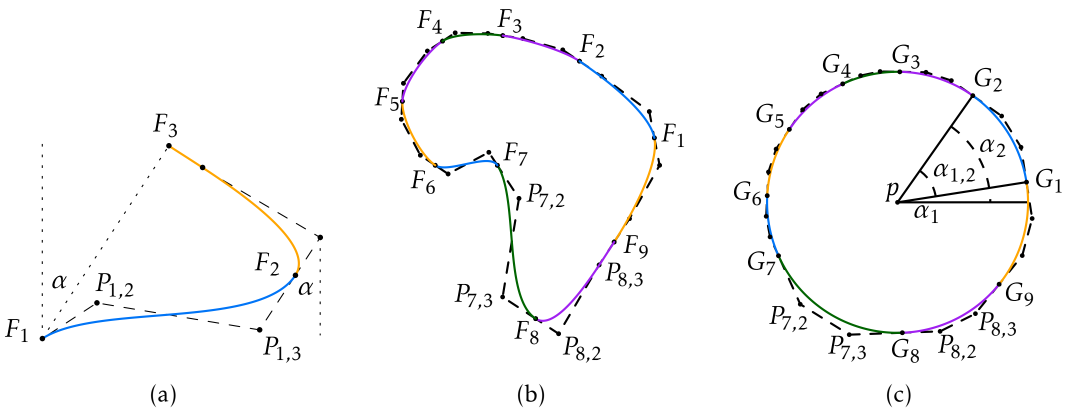

Figure 4.5: (a) The angle of the tangent at F_2 is the angle between F_1 and F_3. The distance between $P_{1,2}$ and F_1, and the distance between $P_{1,3}$ and F_2 is $0.25 \cdot d(F_1, F_2)$. (b) A C^1-continuous Bézier spline. Colored segments denote the individual Bézier curves. The feature points and some control points are labeled. (c) The transformation of the Bézier spline into a circle.

Finally, we set angle of the tangent at feature point F_i to be the angle between F_{i-1} and F_{i+1}, see Figure 4.5(a). This constitutes the information required to draw the star-shaped necklace through the feature points. In the next section we show how we interpolate between a star-shaped curve through the feature points, and a circle.

4.3 Interpolation

We want to be able to smoothly transition between a circular necklace and a star-shaped necklace. This means that we should first convert a circle into a cubic Bézier spline in such a way that the star-shaped necklace and the circular necklace both contain the same number of Bézier curves. Let (p, r) be the representation of the circular necklace, where $p = (x_c, y_c)$ is the center of the circle and r is the radius of the circle.

Transforming a star-shaped spline into a circular spline. In order to create a circular spline through the feature points, we should first move the feature points into a circular configuration. We do this by *unrolling* and scaling the star-shaped necklace we created in the previous section. Unrolling is done as follows. We compute the total length l_s of the star-shaped necklace and we compute the circumference $l_c = 2\pi r$ of the desired circle. This gives us a scaling factor $l_c \,/\, l_s$. We multiply the length of every Bézier curve of the star-shaped necklace by $l_c \,/\, l_s$. Let $l_1, \ldots, l_n$ with $l_1 + \ldots + l_n = l_c$ be the scaled lengths of curves $C_1, \ldots, C_n$. For every feature point F_i, let G_i be the point that is F_i transformed to lie on a circle. The objective is to compute G_i, for all i. Let α_1 be the angle between F_1 and p. We find G_1 by computing $G_1 = (x_c + r \cdot \cos\alpha_1,\ y_c + r \cdot \sin\alpha_1)$. We project the rest of the feature points to (p, r) as follows. Let $\alpha_{1,2} = l_1/r$ be the angle obtained by moving a distance l_1 along a circle with radius r, see Figure 4.5(c). It is the angle between G_1, and the point that will become G_2. To compute G_2, we should know the angle between p and G_2. This angle is $\alpha_2 = \alpha_1 + \alpha_{1,2}$. From this follows that $G_2 = (x_c + r \cdot \cos\alpha_2,\ y_c + r \cdot \sin\alpha_2)$. We repeat this process for all feature points to form $G_1, \ldots, G_n$.

The feature points, that now lie in a circular configuration, represent first and fourth control points of curves in our set of Bézier curves. We should now add second and third control points in such a way that the resulting Bézier spline is a circle. It is impossible to form an exact circular arc using Bézier curves. Goldapp [16] presents various ways to approximate circular arcs using cubic polynomials. The approximation turns out to have sixth order accuracy. Riškus [28] presents a practical method to approximate arbitrary circular arcs using cubic Bézier curves. This method depends on a magic number k. When the circular arc is the upper right arc of a circle, then

$$k = 0.55228...$$

Then, for the unit circle, the corresponding control points are $(1,0)$, $(1,k)$, $(k,1)$, $(0,1)$. Riškus generalizes this for arbitrary circular arcs and for a circle with an arbitrary center and radius. The method by Riškus is as follows. Let $P_1 = (x_1, y_1)$, $P_4 = (x_4, y_4)$ be the first and last control point of a Bézier curve, let $p = (x_c, y_c)$ be the center of the desired circle. Then:

$$\begin{aligned}
x_a &= x_1 - x_c \\
y_a &= y_1 - y_c \\
x_b &= x_4 - x_c \\
y_b &= y_4 - y_c \\
q_1 &= x_a^2 + y_a^2 \\
q_2 &= q_1 + x_a \cdot x_b + y_a \cdot y_b \\
k &= \frac{4}{3}(\sqrt{2 \cdot q_1 \cdot q_2} - q_2) \,/\, (x_a \cdot y_b - x_b \cdot y_a) \\
x_2 &= x_c + x_a - k \cdot y_a \\
y_2 &= y_c + y_a + k \cdot x_a \\
x_3 &= x_c + x_b + k \cdot y_b \\
y_3 &= y_c + y_b - k \cdot x_b
\end{aligned}$$

The remaining control points P_2, P_3 are

$$\begin{aligned}
P_2 &= (x_2, y_2) \\
P_3 &= (x_3, y_3)
\end{aligned}$$

By performing this computation for every pair of consecutive circular feature points G_i, G_{i+1} we create a circular spline. Applying this computation on the curves in Figure 4.5(a) results in the circle shown in Figure 4.5(b). We then linearly interpolate the following properties of the star-shaped spline and the circular spline.

- The positions of the start and endpoints of the Bézier curves.
- The angles of the tangents at the start and endpoints of the Bézier curves.
- The distances between the intermediary control points and the start and endpoints of the Bézier curves.

In Chapter 5 we perform experiments with various interpolation factors. We also show how different circles affect the outcome of the interpolation.

Chapter 5

Experimental Evaluation

In this chapter we experimentally evaluate our algorithms on real maps. Our pipeline works as follows.

1. We convert every polygon of the input map to a set of colored points using the algorithm from Chapter 2. In general we sample every polygon using 20 points, unless mentioned otherwise.

2. We find the smallest disk that contains some fraction of the points of a certain color, for every color, using the algorithm from Chapter 3. In general we use the fraction 1/2, however, in Section 5.1 we will experiment with varying fractions based on the area of the region. For circular necklaces our pipeline stops here. For star-shaped necklaces we continue to step 3.

3. We compute the feature points of the largest polygon of the union of the set of polygons of the input map using the algorithm from Chapter 4.

4. We compute a star-shaped necklace by drawing a Bézier curve through every pair of consecutive feature points using the procedure from Chapter 4. We can then make a choice:

   (a) We interpolate between the circle found in step 2, and the star-shaped necklace found in step 4.

   (b) We interpolate between the smallest enclosing circle of the feature points fount in step 3, and the star-shaped necklace found in step 4.

We give an overview of the experiments we perform. For circular necklaces we first look at a couple of different maps and generate a single circular necklaces for those maps using various fractions for the regions. We then propose a method that sets the fractions according to the area of the regions. We also experiment with ignoring certain regions when computing the necklace. We then experiment with cutting the necklaces for maps in which the area of the regions differs significantly. Finally, we experiment with generating multiple necklaces for a single map.

For star-shaped necklaces we experiment on South America and Africa. We experiment with various interpolation factors and we try out various circles that are used in the interpolation. We also take a look at offsetting the necklaces using Minkowski sums.

Most of the maps are retrieved from the Natural Earth dataset [11], except for the map of the Netherlands, which is hand-drawn. The following Natural Earth maps are used:

- Map `ne_110m_admin_0_countries` is used to generate all maps that show individual countries.

- Map `ne_110m_admin_1_states_provinces` is used to generate the map of the USA.

Hardware and software. The experiments are run on an AMD Ryzen 3600 processor. A prototype of the algorithm that generates circular necklaces has been implemented in Python 3. It takes one second to generate a circular necklace for 240 points (= 14 countries). It takes four seconds to generate a necklace for 400 points, and it takes fifteen seconds to generate a necklace for 760 points. Note that this program runs just on a single core. By using a faster language (e.g. C++) and parallelism, we suspect that the running time could be greatly improved.

For star-shaped necklaces, most of the algorithm has been automated. Using `pyclipper` [27] and GeoPandas [15], we create the union of the set of input polygons and, optionally, inflated versions of the polygons. The medial axis is computed using JavaScript library by Steenkamp [33]. The medial axis is then re-imported into Python 3, where we compute the feature points and the cubic Bézier curves. Once all the parts are chained together, the running time of computing a star-shaped necklace is a few seconds at most, depending on the number of vertices that make up the map. Our code can be found on github.com/Thobro/Necklaces.

5.1 Circular Necklaces

In Figure 5.1 two examples are shown. Both of these examples have been generated by assigning the fraction 1/2 to every region. Even though South America has a specific elongated shape, a circle fits nicely. Since only two countries lie in the southern part of this continent the reader will still be able to associate the symbols with their corresponding regions. For Europe, the necklace is suboptimal: Russia takes up too much of the necklace and Iceland causes the necklace to contain too much water. We will first propose a method that weakens the area requirements for larger polygons.

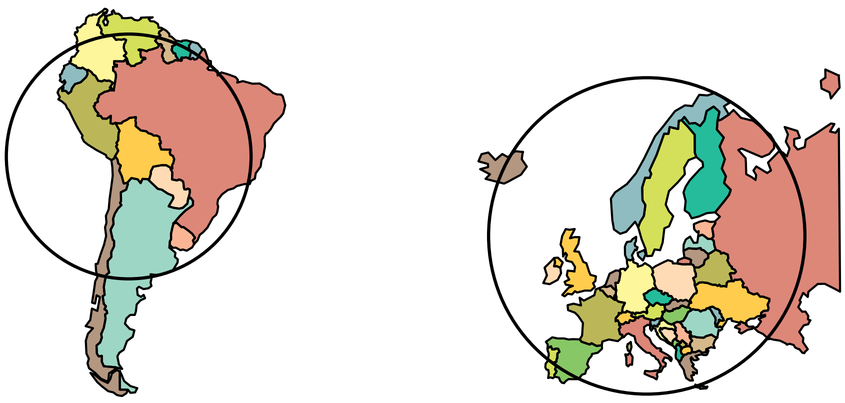
(a) (b)

Figure 5.1: Every country is assigned the fraction 1/2. (a) South America (b) Europe.

Inverted inclusion. We can make better necklaces by weakening the area requirements for larger regions. We propose a method we call *inverted inclusion*. The results of this method can be seen in Figure 5.2. In this figure dashed curves are necklaces generated by assigning every region the fraction 1/2, and solid curves are necklaces generated using inverted inclusion. Inverted inclusion works by giving the largest region a fraction of f_1, and giving the smallest region a fraction of $f_2 \geq f_1$. Every other region is assigned a fraction between f_1 and f_2 dependent on its area. For example, a polygon that is halfway between the largest region and the smallest region in terms of area is assigned the fraction $(f_1 + f_2) / 2$. We say that we apply inverted inclusion in the range of $[f_1, f_2]$. Experiments show that inverted inclusion in [1/8, 1/2] works well. A downside of this method is that the area requirement can only drop in steps of $1/n$, where n is the number of points sampled per region. If enough points are sampled, this should not pose a problem. Another downside is that we do not take the shape of the regions into account. For example, even though Chile (Figure 5.2(a)) seems like a large region visually, inverted inclusion in [1/8, 1/2] gives this region a fraction of 2/5. Inverted inclusion works particularly well when a large region determines the size of the circle, as is the case for Europe—see Figure 5.2(b). In most other cases, inverted occlusion does not produce noticeable difference.

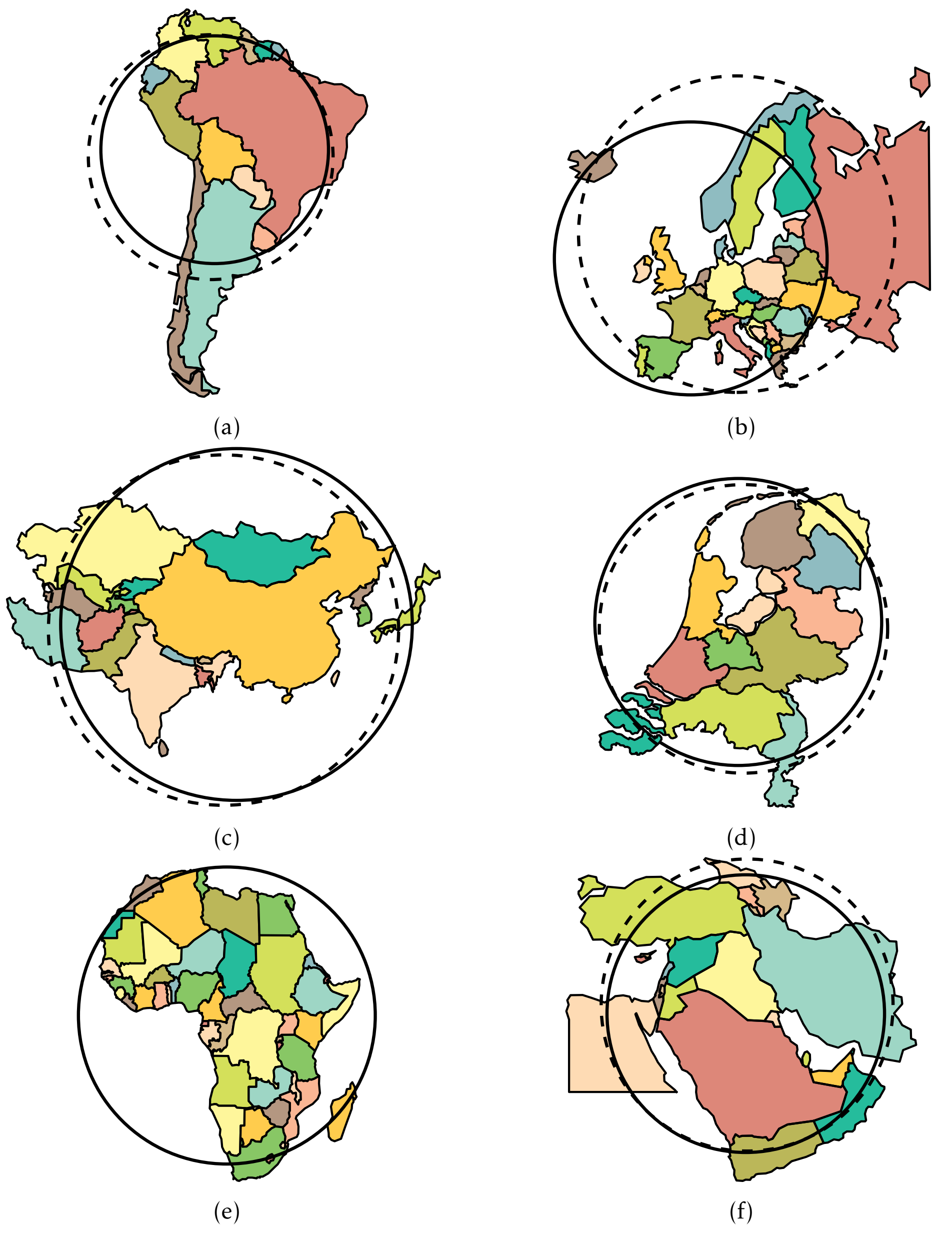

Figure 5.2: Six examples. Countries are sampled using 40 points. The dashed curves represent necklaces found by giving every region a fraction of 1/2. The solid curves represent necklaces found using inverted inclusion in the range of [1/8, 1/2]. (a) South America. (b) Europe. (c) East, South, and Central Asia. (d) The Netherlands. (e) For Africa, using inverted inclusion does not make a difference. (f) The Middle East.

Necklace as the the inner circle of the annulus. In this thesis we have often referred to the curve we generate as *the necklace*. However we should allow some freedom in what the curve actually represents. Depending on the size of the symbols, a symbol can completely occlude a region. We can see this in Figure 5.3(a), where Groningen has a relatively large data value associated to it which makes the symbol occlude the region. To solve this problem we interpret the curve we generate as the inner circle of the annulus created by the symbols and the necklace, as shown in Figure 5.3(b). Note that the symbols are generally not given, and hence we can not decide whether or not we should interpret the curve as the inner circle of the annulus, or as the necklace itself. One could implement a post processing step to the algorithm that generates the symbols. If there are symbols that completely occlude a region, then the symbols should be scaled down in size, or, the radius of the necklace should be increased. We could also add a user-defined parameter that allows the end user to change the radius of the necklace.

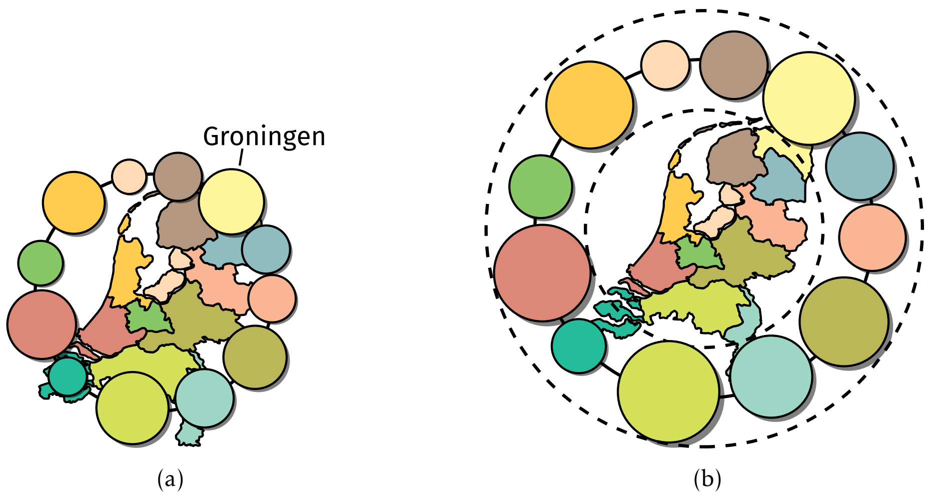

Figure 5.3: The Netherlands with arbitrary beads. (a) We use the generated circle as the necklace. (b) We use the generated circle as the inner circle of the annulus (denoted by the dashed lines).

Discarding regions. In Figure 5.4 Southeast Asia is shown. In this figure the country of East Timor makes the necklace unnecessarily large. Hence, we provide an algorithm that repeatedly discards regions that lie on the boundary of the circle. The objective is to find the smallest disk d that contains at most a fraction f_w of the water, and satisfies a maximum number of region fractions. We do this by modeling the water as a set of points of color w called *water points*. This set of points is sampled using the techniques from Chapter 2. We require that the number of water points that lie in d is at most $c_w = f_w \cdot w$, where w is the number of water points. We provide a heuristic approach to satisfy this constraint. We start with $d = d^*$, where d^* is the smallest colored k-enclosing disk. We then iteratively discard the region that gives the largest decrease in the number of water points that lie in d. This means that we have to repeatedly execute the algorithm from Chapter 3 on a shrinking set of regions until no more than c_w points of color w lie in d. At every iteration, the region we discard is one of the at most three regions that determine d. Thus, the total running time of this greedy algorithm is $O(mn^2 \log n)$, where n is the total number of points we sample from the map and m is the number of regions.

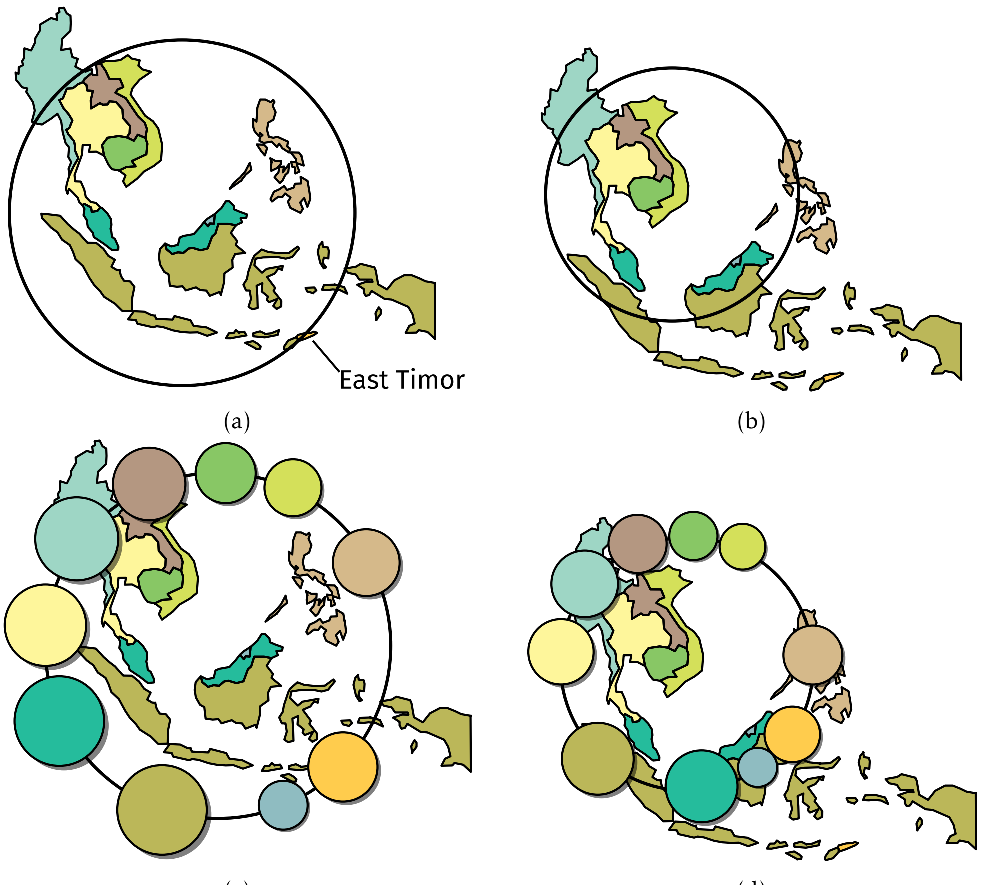

Figure 5.4: Southeast Asia. Inverted inclusion in the range of [1/8, 1/2]. (a) Including all countries. (b) Excluding East Timor. (c, d) Both necklaces with arbitrary beads shown.

In Figure 5.5 a few experimental results are shown for Europe, for various values of c_w. Of course, a sensible choice of c_w differs per map. Southeast Asia is a region that already contains a large amount of water between the countries and thus will always require a greater value of c_w compared to, say, South America. A heuristic approach also does not always work well. Imagine Iceland being split into two regions A and B, where A is small and further removed from the center of d^* than B. Our heuristic algorithm will likely never discard A, since this would result in a circle that is only slightly smaller. This is a problem because discarding first A and then B would result in a much smaller circle, as we have seen in Figure 5.5. A better method is required to adequately reduce the amount of water that lies within the necklace, however, this experiment does show the flexibility of the algorithm we presented in Chapter 3.

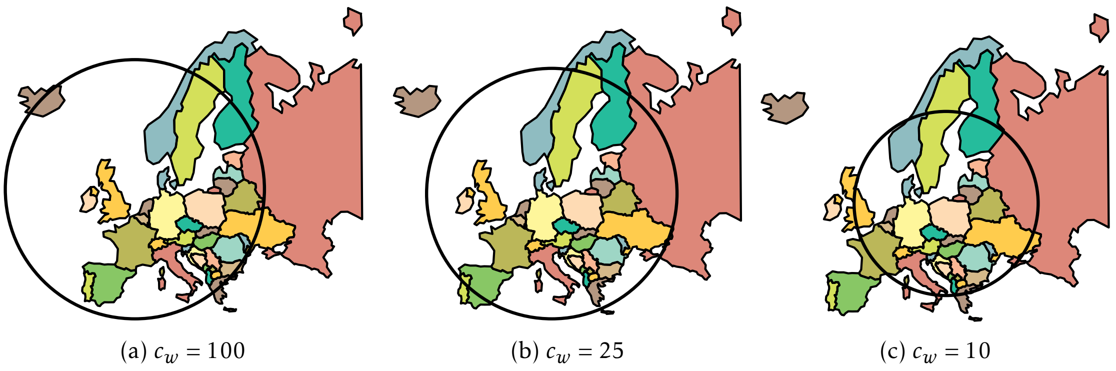
(a) $c_w = 100$ (b) $c_w = 25$ (c) $c_w = 10$

Figure 5.5: Excluding water points for various values of c_w. Inverted inclusion in [1/2, 1/8]. The water has been sampled using 100 points and every country has been sampled using 20 points.

Incomplete circles. In Figure 5.6 North America and Central America are shown. North America consists of a few large countries whereas Central America consist of small countries. In Figure 5.6(a) a necklace is shown that has been generated using inverted inclusion in [1/8, 1/2]. While this method already causes Canada to lie mostly outside of the necklace, a large part of the necklace is empty. We could use the previous method to ignore Canada when computing the necklace but this would not solve our issue. The necklace would become too small in that case. Instead, we solve this issue by cutting the circle into a circular arc, as shown in Figure 5.6(b). Such an arc is a valid necklace and is better suited for this set of regions. While our current algorithms do not generate circular arcs, one could extend the algorithm from Chapter 3 to produce circular arcs.

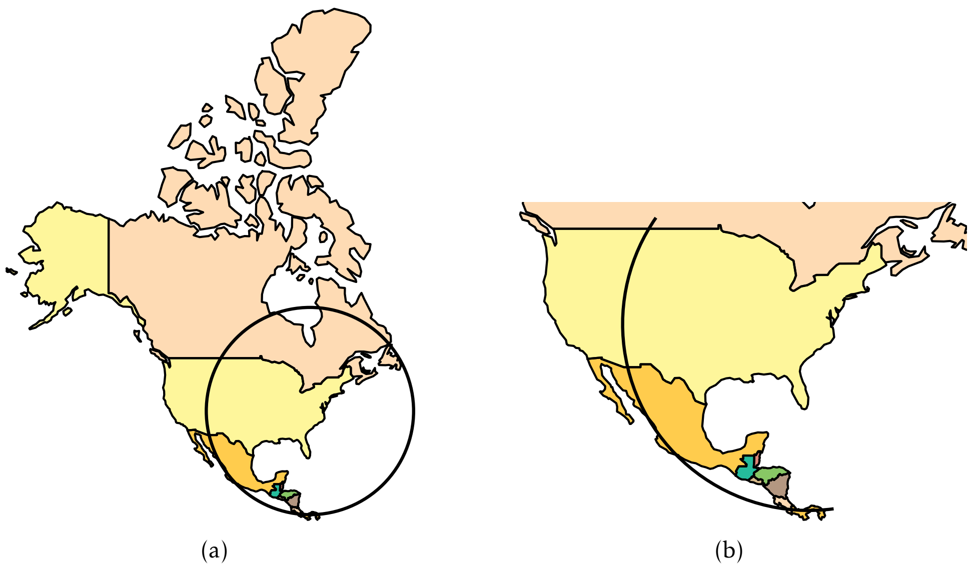
(a) (b)

Figure 5.6: Northern America and Central America. (a) A combination of large and small countries does not always yield a good looking circle. (b) A small section of the circle.

Disjoint circles. Another issue that can occur is that there may be too many regions within the necklace. This makes it hard for the reader to associate symbols to regions that lie far away from the necklace—in Figure 5.7 the USA is shown with a single necklace.

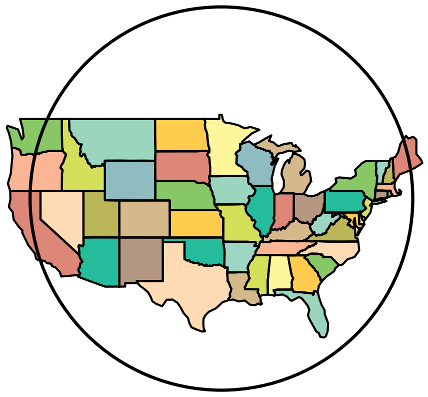

Figure 5.7: The USA. A problem with this map is that there are too many regions within the necklace.

As was also proposed by Speckmann and Verbeek [31], we can partition the set of regions into smaller sets and compute a necklace for every partition. An example of this is shown in Figure 5.8(a). In this figure, the states of the USA are partitioned into four sets based on the major regions of the USA the US Census Bureau considers. However, there is a high probability that these circles overlap, as seen in Figure 5.8(a). Here we can again cut the circles into circular arcs in order to mitigate this issue. A result is shown in Figure 5.8(b).

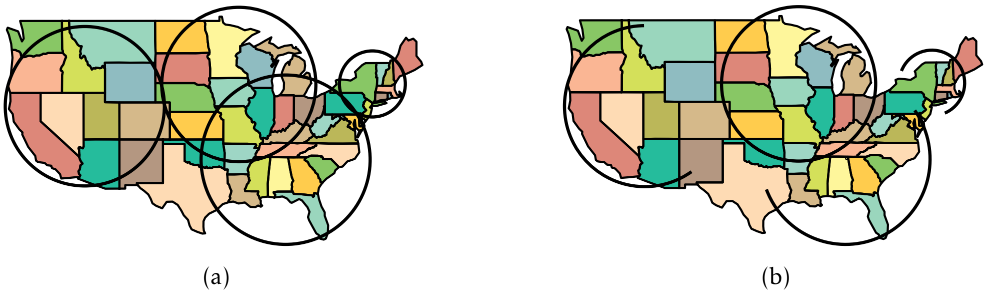

Figure 5.8: The USA split based on the four regions the US Census Bureau considers. (a) Using circles. (b) Using circular arcs.

5.2 Star-shaped Necklaces

For some maps a star-shaped necklace is a better option, as we have seen in the introduction. We test our algorithm that generates star-shaped necklaces on South America and Africa. As we will see, star-shaped necklaces work well for these continents. Let us first take a look at South America. Recall that we have two options for interpolation between circular necklaces and star-shaped necklaces. We can interpolate between the star-shaped necklace found using the procedure from Chapter 4, and the smallest colored k enclosing disk found using the algorithm from Chapter 3. We can also interpolate between the star-shaped necklace and the smallest enclosing disk of the feature points. Let us first take a look at the first option, of which the output can be seen in Figure 5.9.

As input we take the polygons that represent the regions of the map. We compute the corresponding smallest colored k-enclosing disk using the Algorithm from Chapter 3. We then compute the star-shaped necklace using the algorithm from Chapter 4. Finally, we interpolate between the results of these algorithms. An interpolation factor of 0 is a circle, and an interpolation factor of 1 is the generated star-shaped necklace. In Figure 5.9(a)–(d) we show the output of these algorithms for various interpolation factors.

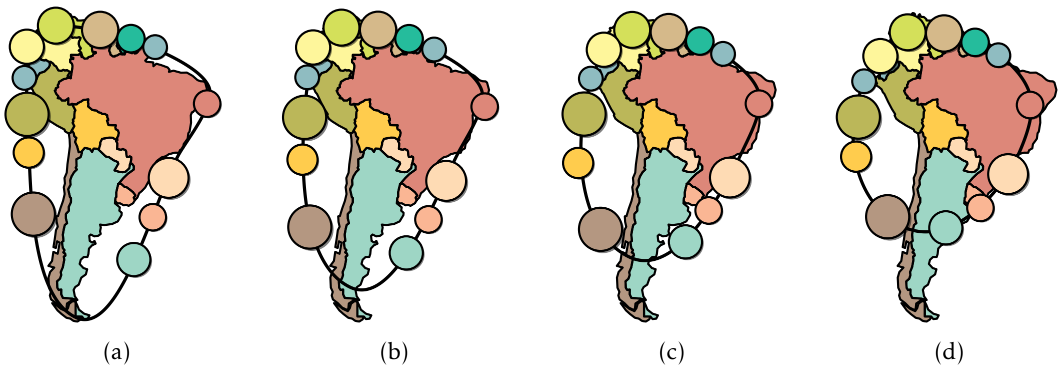

Figure 5.9: South America. Interpolation between the circle generated using the algorithm from Chapter 3, and the generated star-shaped necklace. For (a)–(d) the interpolation factors are 1, 0.75, 0.5, 0.25 respectively. Arbitrary symbols are placed to simulate how the necklace map could look like.

In Figure 5.10 we interpolate between the generated star-shaped necklace, and the smallest enclosing circle of the feature points. Since we use the smallest enclosing circle of the feature points, we have that the necklace represents the shape of the continent more closely for the various levels of interpolation. However, it can also result in a shape that is too large, as seen in Figure 5.10(d).

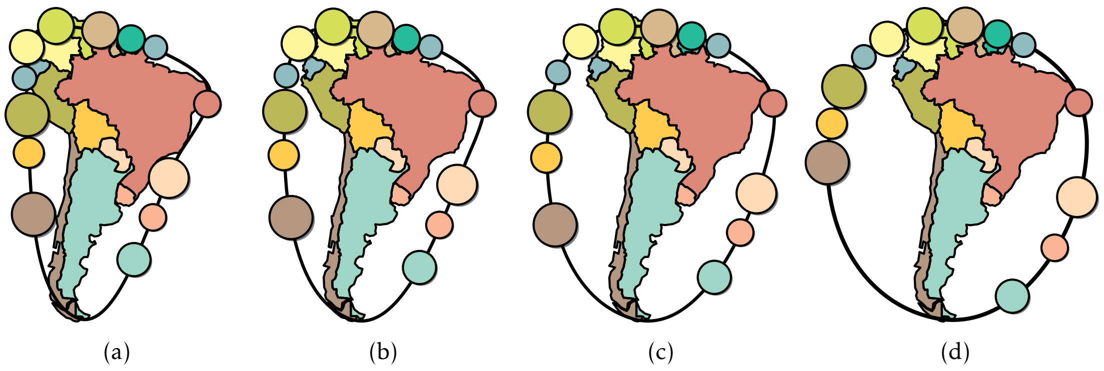

Figure 5.10: South America. Interpolation between the smallest enclosing circle of the feature points, and the generated star-shaped necklace. For (a)–(d) the interpolation factors are 1, 0.75, 0.5, 0.25 respectively. Arbitrary symbols are placed to simulate how the necklace map could look.

In Figure 5.11 Africa is shown. For this continent the circle computed using the algorithm from Chapter 3 is nearly identical to the smallest enclosing circle of the feature points. Hence, we show only an interpolation using the circle computed using the algorithm from Chapter 3. In Figure 5.11(e)–(f) the necklace leaves too much empty space, similar to Figure 5.10(d). Hence, we conclude that a suitable interpolation factor is at least, say, 0.5.

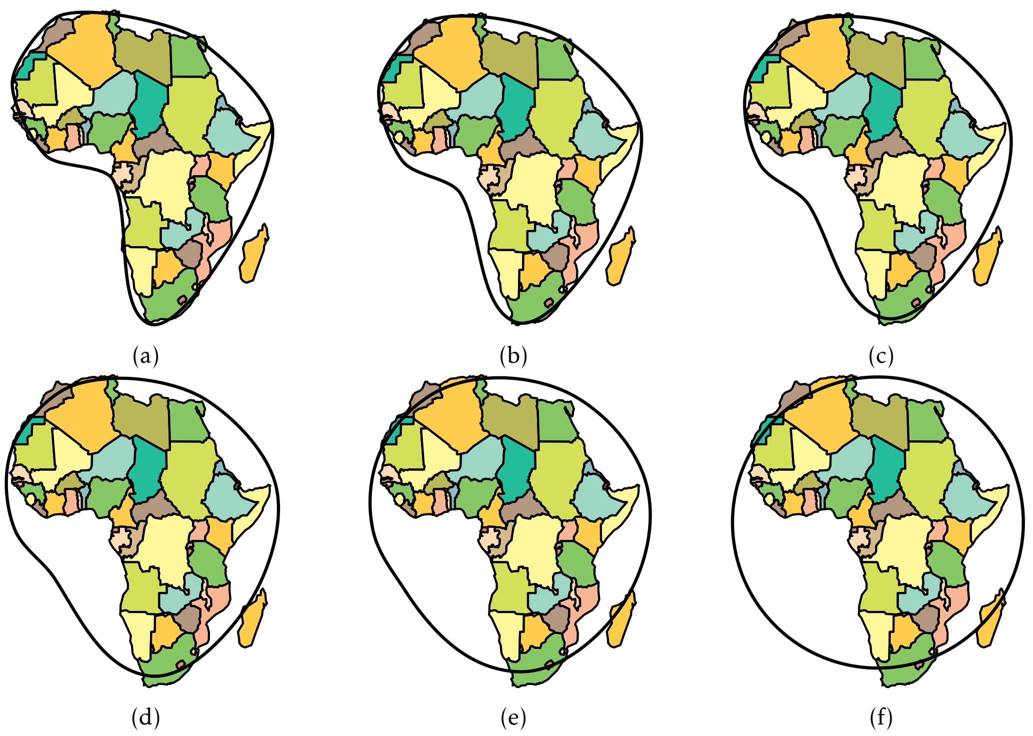

Figure 5.11: Africa. Interpolation between the circle generated using the algorithm from Chapter 3, and the generated star-shaped necklace. For (a)–(f) the interpolation factors are 1, 0.8, 0.6, 0.4, 0.2, 0 respectively.

Region offsets. In Figures 5.9, 5.10, and 5.11 the necklace lies very close to the regions. This can cause a symbol to completely occlude a region, as can be seen in Figure 5.9(a). We can solve this problem by computing the necklace on a larger version of the input polygon, see Figure 5.12. We first inflate the input polygon (Figure 5.12(a)). Then, we compute the feature points for this inflated polygon and compute the corresponding star-shaped necklace (Figure 5.12(b)). Finally, we use this necklace as a necklace for the original polygon (Figure 5.12(c)).

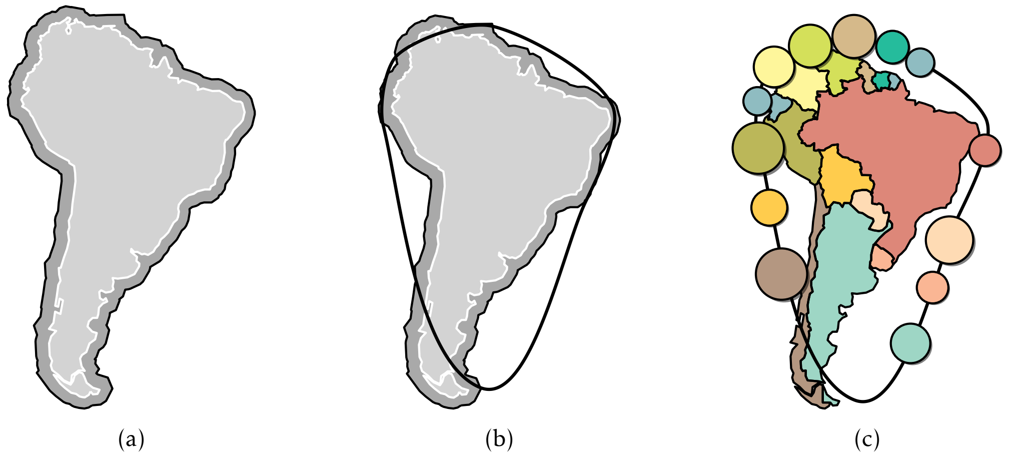

Figure 5.12: (a) A polygon in light gray, together with the inflated version of the same polygon in dark gray. The inflated version is constructed by computing the Minkowsi sum of the polygon with a square. (b) A necklace computed from the inflated polygon. (c) The generated necklace used as a necklace for the original polygon. Arbitrary beads are shown.

We compute the inflated polygon using *Minkowski sums*. The Minkowski sum of two sets of points A, B is

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

Let P be the input polygon and let S be a square of a specific width. Then the Minkowski sum of P and S results in the polygon shown in Figure 5.12(a). Typically, polygon offsets are computed using disks in the Minkowski sum instead of squares. However, this would produce circular arcs in our polygon. In Figure 5.13 we see that a polygon that contains circular arcs also contains edges in its medial axis that stop before they reach the boundary of the polygon. Since we take feature points from edges of the medial axis that coincide with the boundary of the polygon, we can not extract feature points from edges that are directed into a circular arc.

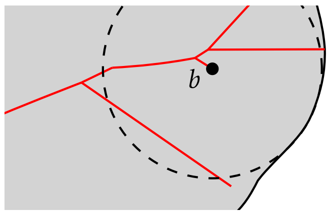

Figure 5.13: The edge denoted by b does not continue until it reaches the boundary of the polygon.

The use of Minkowski sums allows us to enlarge the necklace in such a way that the beads do not occlude the regions too much. However, a downside is that Minkowski sums can distort the shape of the polygon. As a result, the generated necklace might not accurately capture the shape of the continent anymore. For example, the southern tip of the inflated version of South America (Figure 5.12(a)) causes the bottom tip of necklace to move too far to the right.

We conclude that our algorithms produce visually pleasing results. Inverted inclusion is a technique that works well for all sorts of maps. For maps that contain many regions or for maps in which there is a large size difference between the regions, our algorithm produces results that are not as good. However, as we alluded to before, extensions could be developed that, for example, cut the circular necklace into circular arcs. The star-shaped necklaces we produce work well and are also flexible. One can simply discard individual Bézier curves from the star-shaped spline in order to create disjoint necklaces. Computing an offset star-shaped necklace using a Minkowski sum is not ideal as distortions can occur. Hence, for this a better method should be used.

Chapter 6

Conclusions

In this thesis we have presented a few methods that successfully generate good necklaces for necklace maps. We presented an algorithm that computes a suitable circular necklace given a set of regions. This algorithm works by first approximating the regions as a set of colored points, and then computing the smallest disk that contains at least some number of points of a certain color, for every color. Our algorithm finds this disk in expected $O(n^2 \log n)$ time, where n is the total number of points. We determine the color requirements by giving large regions a smaller color requirement, and small regions a larger color requirement. We also presented an algorithm that generates suitable star-shaped necklaces. This algorithm computes a small set of feature points using the medial axis, that describe the shape of the continent, and then draws a cubic Bézier spline through the feature points. To provide flexibility we presented a method that interpolates between circular necklaces and star-shaped necklaces.

In Chapter 5 we concluded that our algorithms are flexible. Our circular necklaces allow cuts to be made in order to form circular arc necklaces. Our algorithm can also be used to generate multiple necklaces for a single map. For star-shaped necklaces we are able to discard individual Bézier curves in order to form, possibly multiple, open necklaces.

Future work. The expected running time of $O(n^2 \log n)$ for finding the smallest colored k-enclosing disk, can be improved to $O(n^2 \log m)$, if there was a way to solve the problem in $O(n^2)$ time for a set of points of at most four colors. However, we are not sure if this is possible.

In our algorithm a couple of user-defined parameters have been defined. One such parameter, γ, dictates the minimum distance between any pair of consecutive feature points. Ideally this parameter should be chosen automatically. Another user-defined parameter determines whether the generated curve represents the inner circle of the annulus, or the necklace itself. It is difficult to automate this choice, as this choice depends on the size of the symbols, something which is unknown to the algorithms.

In Chapter 5 we have seen that circular necklaces do not work well if there is a large size difference between the regions. We proposed a method that cuts the circular necklace into circular arc necklaces. Future work should look at how to automatically make this cut. Better results can likely be produced by developing a separate algorithm specifically for open necklaces. We also experimented with ignoring regions when computing the necklace. While this experiment produced good preliminary results, a more algorithmic approach to this problem is desired.

Lastly, star-shaped necklaces can lie too close to their corresponding continent. We proposed a method that computes a star-shaped necklace for the inflated version of a continent, where inflation is done using Minkowski sums. However, this method might introduce distortions that cause the shape of the star-shaped necklace to misalign with the shape of the non-inflated version of the continent. Hence, another method that offsets the feature points, could potentially provide better results.

Bibliography

[1] Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. The farthest color voronoi diagram and related problems. In *Abstracts 17th European Workshop Comput. Geom*, pages 113–116, 2001. 6

[2] Pankaj K Agarwal, Mark De Berg, Jirí Matousek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM journal on computing*, 27(3):654–667, 1998. 6

[3] Xiang Bai and Longin Jan Latecki. Discrete skeleton evolution. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 362–374. Springer, 2007. 24

[4] Luis Barba, Stephane Durocher, Robert Fraser, Fernando Alfredo Hurtado Díaz, Saeed Mehrabi, Debajyoti Mondal, Jason Morrison Morrison, Matthew Skala, and Mohammad Abdul Wahid. On k-enclosing objects in a coloured point set. In *Proceedings of the 25th Canadian Conference on Computational Geometry*, pages 229–234, 2014. 6

[5] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. 8

[6] Harry Blum et al. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967. 23

[7] Sergio Cabello, Herman Haverkort, Marc Van Kreveld, and Bettina Speckmann. Algorithmic aspects of proportional symbol maps. *Algorithmica*, 58(3):543–565, 2010. 5

[8] Francis Chin, Jack Snoeyink, and Cao An Wang. Finding the medial axis of a simple polygon in linear time. *Discrete & Computational Geometry*, 21(3):405–420, 1999. 24

[9] Paul De Casteljau. Outillages méthodes calcul. *Andr e Citro en Automobiles SA, Paris*, 1959. 4, 25

[10] Daniel Dorling. Area cartograms: their use and creation. *Concepts and Techniques in Modern Geography (CATMOG)*, 1996. 5

[11] Natural Earth. Natural Earth homepage. `https://www.naturalearthdata.com/`. [Online; accessed 9-July-2020]. 29

[12] Alon Efrat, Micha Sharir, and Alon Ziv. Computing the smallest k-enclosing circle and related problems. *Computational Geometry*, 4(3):119–136, 1994. 5, 16

[13] Martin Fink, Herman Haverkort, Martin Nöllenburg, Maxwell Roberts, Julian Schuhmann, and Alexander Wolff. Drawing metro maps using bézier curves. In *International Symposium on Graph Drawing*, pages 463–474. Springer, 2012. 6

[14] Michael R Garey, David S Johnson, Franco P Preparata, and Robert E Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175–179, 1978. 10

[15] GeoPandas. Geopandas. https://geopandas.org/. [Online; accessed 9-July-2020]. 30

[16] Michael Goldapp. Approximation of circular arcs by cubic polynomials. *Computer Aided Geometric Design*, 8(3):227–238, 1991. 26

[17] Jacob E Goodman, János Pach, Pach János, and Emo Welzl. *Combinatorial and computational geometry*, volume 52. Cambridge University Press, 2005. 15

[18] Sariel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005. 6, 13, 15

[19] David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987. 8

[20] Daniel P Huttenlocher, Klara Kedem, and Micha Sharir. The upper envelope of voronoi surfaces and its applications. *Discrete & Computational Geometry*, 9(3):267–291, 1993. 6

[21] Jiří Matoušek. On enclosing k points by a circle. *Information Processing Letters*, 53(4):217–221, 1995. 6, 16, 17

[22] Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4-5):498–516, 1996. 15

[23] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30(4):852–865, 1983. 18

[24] Nimrod Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM journal on computing*, 12(4):759–776, 1983. 5, 18

[25] Xiaofeng Mi, Doug DeCarlo, and Matthew Stone. Abstraction of 2d shapes in terms of parts. In *Proceedings of the 7th international symposium on non-photorealistic animation and rendering*, pages 15–24, 2009. 6

[26] Nabil H Mustafa and Kasturi R Varadarajan. Epsilon-approximations and epsilon-nets. *arXiv preprint arXiv:1702.03676*, 2017. 9

[27] Pyclipper. Pyclipper. https://github.com/fonttools/pyclipper. [Online; accessed 9-July-2020]. 30

[28] Aleksas Riškus. Approximation of a cubic bézier curve by circular arcs and vice versa. *Information Technology and control*, 35(4), 2006. 26

[29] Doron Shaked and Alfred M Bruckstein. Pruning medial axes. *Computer vision and image understanding*, 69(2):156–169, 1998. 24

[30] Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 567–579. Springer, 1992. 14

[31] Bettina Speckmann and Kevin Verbeek. Necklace maps. *IEEE Transactions on Visualization & Computer Graphics*, 16(6):881–889, 2010. iii, 1, 5, 35

[32] Bettina Speckmann and Kevin Verbeek. Algorithms for necklace maps. *International Journal of Computational Geometry & Applications*, 25(01):15–36, 2015. 2, 3, 4, 5

[33] Floris Steenkamp. Medial (and scale) axis transform library - SVG focused. https://github.com/FlorisSteenkamp/MAT. [Online; accessed 9-July-2020]. 30

[34] Waldo R Tobler. Pseudo-cartograms. *The American Cartographer*, 13(1):43–50, 1986. 5

[35] Arthur van Goethem, Wouter Meulemans, Andreas Reimer, Herman Haverkort, and Bettina Speckmann. Topologically safe curved schematization. *The Cartographic Journal*, 50(3):276–285, 2013. 6

[36] Arthur van Goethem, Wouter Meulemans, Bettina Speckmann, and Jo Wood. Exploring curved schematization. In *2014 IEEE Pacific Visualization Symposium*, pages 1–8. IEEE, 2014. 6

[37] Marc Van Kreveld and Bettina Speckmann. On rectangular cartograms. *Computational Geometry*, 37(3):175–187, 2007. 5

[38] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. 8

[39] VN Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264, 1971. 8, 9

[40] John K Wright. Problems in population mapping. *Notes on Statistical Mapping With Special Reference to the Mapping of Population Phenomena*, pages 1–18, 1938. 5